Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería en Informática

Proyecto Fin de Carrera

Desarrollo de una interfaz en Silverlight para la gestión de un sistema de colas en clusters

Autor: Rafael Montero Montero

Tutor: D. Francisco Javier García Blas

Noviembre, 2009

[Página dejada intencionadamente en blanco]

AGRADECIMIENTOS

Agradecer toda la ayuda prestada a lo largo del proyecto al tutor del mismo, Javi, que a pesar de estar muy ocupado, de una manera u otra siempre ha estado ahí. Ánimo con la tesis.

También agradecer a todas las personas que me han demostrado su apoyo durante todos estos años de estudio, especialmente a toda mi familia y a mis amigos.

"No creo que Dios quiera exactamente que seamos felices, quiere que seamos capaces de amar y de ser amados, quiere que maduremos, y yo sugiero que precisamente porque Dios nos ama nos concedió el don de sufrir; o por decirlo de otro modo: el dolor es el megáfono que Dios utiliza para despertar a un mundo de sordos; porque somos como bloques de piedra, a partir de los cuales el escultor poco a poco va formando la figura de un hombre, los golpes de su cincel que tanto daño nos hacen también nos hacen más perfectos".

C. S. Lewis

Dedico este proyecto a mis padres por creer en mí en todo momento y ayudarme a hacer posible que haya llegado hasta aquí.

Rafa M.M.

Contenido

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

[Página dejada intencionadamente en blanco]

"Honeste vivere, naeminem laedere et jus sum cuique tribuere"
Ulpiano

1. INTRODUCCIÓN

A continuación, se expone la motivación y objetivos de este proyecto fin de carrera. Además se incluirá los términos y definiciones relacionadas con el mismo, así como la estructura del presente documento.

1.1. MOTIVACIÓN

Un gestor de trabajos (o gestor de colas) es un sistema de distribución y ejecución de trabajos batch, es decir, programas que se ejecutan de modo desatendido, sin interactuar con el usuario. Su misión consiste en ejecutar trabajos aprovechando al máximo los recursos del cluster y garantizando el uso equitativo del sistema para todos los usuarios.

En determinadas ocasiones, acceder a un sistema de colas para ejecutar programas no resulta una tarea sencilla. Muchas veces pueder resultar engorroso abrir termiales remotos, etc. Por este motivo, disponer de una interfaz web a la cual se pueda acceder desde cualquier equipo conectado a la red y nos permita gestionar los trabajos enviados pudiendo ser accedida por diferentes usuarios al mismo tiempo, resulta un complemento sencillo y atractivo.

Otro de los problemas que cuentan los sistemas de colas, es que notifican la terminación de los trabajos mediante correo electrónico, pero no notifican más información que podría ser util para los usuarios como el tiempo de ejecución, estado, resultados, etc.

1.2. OBJETIVOS

Se pretende desarrollar una interfaz en Silverlight para la gestión de un sistema de colas en clusters. Dicho sistema consiste en una aplicación cliente y su correspondiente servidor. El servidor se ha desarrollado en Java en un entorno Linux, acompañado de una herramienta para la ejecución de trabajos en colas llamada *Torque [9]*. El cliente, por su parte, se ha desarrollado con la tecnología *Silverlight* para facilitar el acceso remoto vía Web. A continuación se describirán algunas funciones que serán posibles realizar desde la aplicación:

- Enviar trabajos con sus correspondientes parámetros para que sean ejecutados en la cola indicada.
- Acceder a la información de los trabajos enviados y las salidas y errores devueltos por éstos.
- Visualizar información sobre las colas existentes en el servidor.
- Información sobre los nodos que componen el cluster.
- Acceder al historial de trabajos ejecutados para poder editar sus parámetros de ejecución o volver a enviarlos.

1.3. ESTRUCTURA DEL DOCUMENTO

A continuación se describirán los apartados en los que consta el presente documento, para así tener una visión general del mismo.

En el apartado 1, *Introducción*, se proporciona una visión general del proyecto mediante una breve explicación, incluyendo también características y definiciones que surgirán en el documento.

En el apartado 2, *Estado del arte,* se incluye información detallada sobre las diferentes tecnologías utilizadas para la realización del Proyecto.

En el apartado 3, *Análisis del Proyecto,* se realiza una descomposición del problema a abordar, de esta manera se consigue más claridad a la hora de diseñar el programa y su correspondiente implementación.

En el apartado 4, *Diseño del Proyecto,* se detallan los componentes del Sistema partiendo del análisis antes mencionado. Este apartado será imprescindible para lograr una buena codificación del Sistema.

En el apartado 5, *Pruebas,* se expondrán una serie de pruebas elegidas para verificar que el Sistema realiza correctamente todas las funciones descritas en anteriores apartados.

En el apartado 6, *Conclusiones y trabajos futuros,* se realizará un análisis de los objetivos logrados en el proyecto y se expondrán algunas conclusiones obtenidas del trabajo realizado. Finalmente se comentarán los diferentes proyectos que puedan relacionados con el mismo.

En el apartado 7, *Bibliografía,* se incluirán todas las fuentes de información que se han tenido en cuenta a la hora de desarrollar el proyecto.

Para terminar, se incluyen dos apéndices: *Requisitos mínimos* y *Manual de usuario.* El primero detalla las características mínimas necesarias para los componentes Sistema. El segundo es una breve guía de usuario para el uso del Sistema.

1.4. CARACTERÍSTICAS DEL DOCUMENTO

A continuación se detallarán las características de formato del presente documento:

- Las palabras que no correspondan al lenguaje castellano se mostrarán en letra cursiva.
- La fuente *Times New Roman* es la utilizada para los títulos. El tamaño de los títulos dependerá del nivel de anidamiento, siendo éstos: título 1 en tamaño 18, título 2 en tamaño 16 y título 3 en tamaño 14.
- La fuente *Times New Roman* se utiliza para el resto de texto del documento. El tamaño de letra es 12 y con un interlineado de 1,5.

1.5. DEFINICIONES Y ACRÓNIMOS

1.5.1. DEFINICIONES

- **Web** → Red informática, especialmente para referirse a Internet.
- **Silverlight** → Es un complemento para navegadores de Internet basado en la plataforma Windows que agrega nuevas funciones multimedia como la reproducción de vídeos, gráficos vectoriales, animaciones y de entorno de desarrollo.
- **Torque** → Gestiona los trabajos por lotes (*batch*). Es un producto *OpenSource*, basado en el gestor de trabajos por lotes *OpenPBS*
- **Web Service** → Es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones
- **Interfaz Web** → Aquella aplicación que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador
- **Cluster** → Se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora
- **Tarea/Trabajo** → Programa informático que realiza un cálculo o tarea determinada.
- **Nodo** → Elemento del cual está formado el cluster. El nodo puede tener dos estados: Ocupado o Libre.

1.5.2. ACRÓNIMOS

ACRÓNIMO	SIGNIFICADO
HTTP	Hypertext Transfer Protocol
PAT	*XML Parse Toolkit*. Herramienta de reconocimiento de XML
RPC	*Remote Procedure Call*. Llamada de procedimientos remotos
XAML	*eXtensible Application Markup Language*

SOAP	*Simple Object Access Protocol*
UDDI	*Universal Description Discovery and Integration*
WSDL	*Web Services Definition Service*
XML	*Extensible Markup Language*. Lenguaje de marcado extensible
PBS	*Portable Batch*
API	*Application Program* Interface. Interfaz de aplicación
DOM	*Document Object Model*
GNU	*GNU is Not Unix*
SDK	Software development kit
CSS/DHTML	Cascading Style Sheets
LINQ	Language-Integrated Query
JAVA RMI	Java Remote Method Invocation
SAML	Security Assertion Mark-up Language
SRGC	Sistema Remoto de Gestión de Colas

Tabla 1: Acrónimos

"La honestidad no es una virtud, es una obligación"
A. Calamaro

2. ESTADO DEL ARTE

En este apartado entraremos en detalle en cada una de las tecnologías utilizadas en el desarrollo de este proyecto fin de carrera.

2.1. PLATAFORMA .NET

2.1.1. Introducción a .NET

.NET [1] es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. .NET se podría considerar como una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, en competencia con la plataforma Java de Sun.

A largo plazo Microsoft pretende reemplazar la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) Win32 o Windows API con la plataforma .NET. Esto debido a que la API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows. La plataforma .NET pretende solventar la mayoría de estos

problemas proveyendo un conjunto único y expandible con facilidad, de bloques interconectados, diseñados de forma uniforme y bien documentados, que permitan a los desarrolladores tener a mano todo lo que necesitan para producir aplicaciones sólidas.

Debido a las ventajas que la disponibilidad de una plataforma de este tipo puede darle a las empresas de tecnología y al público en general, muchas otras empresas e instituciones se han unido a Microsoft en el desarrollo y fortalecimiento de la plataforma .Net, ya sea por medio de la implementación de la plataforma para otros sistemas operativos aparte de Windows (Proyecto Mono de Ximian/Novell para Linux/MacOS X/BSD/Solaris), el desarrollo de lenguajes de programación adicionales para la plataforma (ANSI C de la Universidad de Princeton, NetCOBOL de Fujitsu, Delphi de Borland, entre otros) o la creación de bloques adicionales para la plataforma (como controles, componentes y librerías de clases adicionales); siendo algunas de ellas iniciativas de distribución gratuita bajo la licencia GNU.

Actualmente, el Framework de .Net es una plataforma no incluida en los diferentes sistemas operativos distribuidos por Microsoft, por lo que es necesaria su instalación previa a la ejecución de programas creados mediante .Net. El Framework se puede descargar gratuitamente desde la Web oficial de Microsoft.

2.1.2. La arquitectura .NET Framework

Es la arquitectura básica de la plataforma .Net. El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución distribuido.

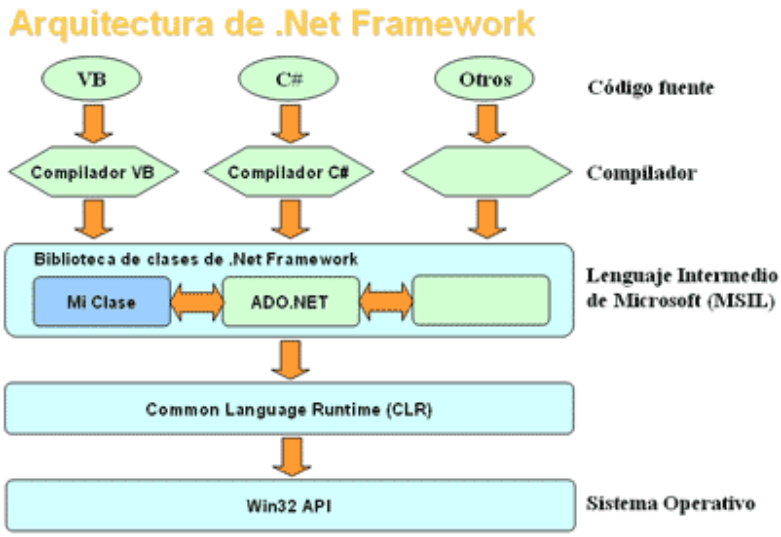


Ilustración 1: Arquitectura .NET

Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (*Common Language Runtime*)

.Net Framework soporta múltiples lenguajes de programación y, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Lenguajes incluidos en la plataforma: C# (C *Sharp*) [2], Visual Basic, C++, J# (Java #), etc.

2.1.3. Common Language Runtime

El CLR es el verdadero núcleo del Framework de .Net, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .Net en un código intermedio (MSIL, Microsoft Intermediate Lenguaje), similar al BYTECODE de Java. Para generar dicho código el compilador se basa en el Common Language Specification (CLS) que determina las reglas necesarias para crear ese código MSIL compatible con el CLR.

Para ejecutarse se necesita un segundo paso, un compilador JIT (Just-In-Time) es el que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .Net independencia de la plataforma hardware, que no de sistema operativo.

Ilustración 2: Rutime de Lenguaje Común

La compilación JIT la realiza el CLR a medida que el programa invoca métodos, el código ejecutable obtenido, se almacena en la memoria caché del ordenador, siendo recompilado de nuevo sólo en el caso de producirse algún cambio en el código fuente.

2.1.4. Bibliotecas de clases

Las clases base proporcionan funciones estándar, como las de entrada/salida, manipulación de cadenas, administración de seguridad, comunicaciones en red, administración de subprocesos, administración de textos y funciones de diseño de la interfaz de usuario.

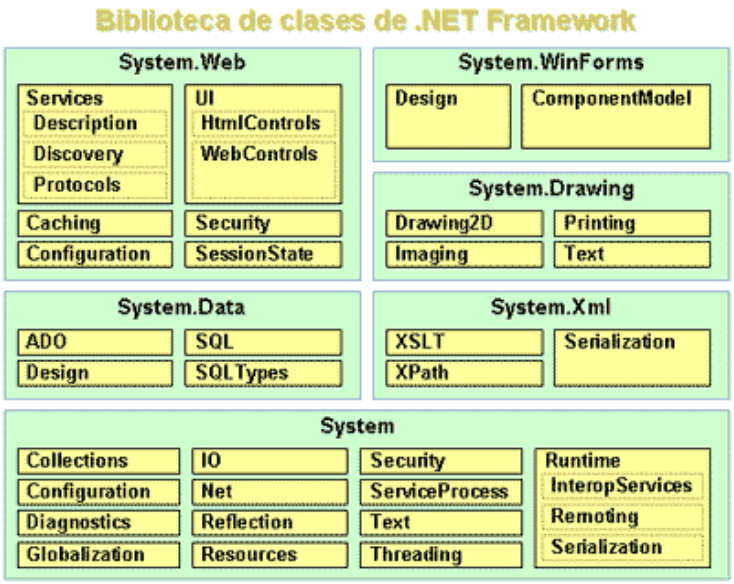


Ilustración 3: Biblioteca de clases de .Net Framework

Las clases de ADO.NET permiten a los programadores interactuar con los datos obtenidos en formato XML a través de las interfaces OLE DB, ODBC, Oracle y SQL Server. Las clases XML permiten la manipulación, búsqueda y conversión de objetos XML. Las clases ASP.NET son compatibles con el desarrollo de aplicaciones basadas en Web y de servicios Web XML. Las clases de *Windows Forms* son compatibles con la generación de aplicaciones cliente inteligentes basadas en escritorio. En conjunto, las bibliotecas de clases ofrecen una interfaz de desarrollo común y coherente en todos los lenguajes compatibles con Windows .NET Framework.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (*namespaces*), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es obtener información de una fuente de datos se utilizará el espacio de nombres System.Data.

La principal ventaja de los espacios de nombres de .Net es que de esta forma se tiene toda la biblioteca de clases de .Net centralizada bajo el mismo espacio de nombres (System).

Además, desde cualquier lenguaje se usa la misma sintaxis de invocación, ya que a todos los lenguajes se aplica la misma biblioteca de clases.

2.1.5. Ensamblados

Uno de los mayores problemas de las aplicaciones actuales es que en muchos casos tienen que tratar con diferentes archivos binarios (DLL´s), elementos de registro, conectividad abierta a bases de datos (ODBC), etc.

Para solucionarlo el Framework de .Net maneja un nuevo concepto denominado ensamblado. Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Por tanto la solución al problema puede ser tan fácil como copiar todos los ensamblados en el directorio de la aplicación.

Con los ensamblados ya no es necesario registrar los componentes de la aplicación. Esto se debe a que los ensamblados almacenan dentro de si mismos toda la información necesaria en lo que se denomina el manifiesto del ensamblado. El manifiesto recoge todos los métodos y propiedades en forma de meta-datos junto con otra información descriptiva, como permisos, dependencias, etc.

Para gestionar el uso que hacen la aplicaciones de los ensamblados .Net utiliza la llamada caché global de ensamblados (GAC, *Global Assembly Cache*). Así, .Net Framework puede albergar en el GAC los ensamblados que puedan ser usados por varias aplicaciones e incluso distintas versiones de un mismo ensamblado, algo que no era posible con el anterior modelo COM.

2.1.6. Origen y necesidad de un nuevo lenguaje: C#

C# (leído en inglés “C Sharp” y en español “C Almohadilla”) es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET.

La sintaxis y estructuración de C# es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son comparables con los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el .NET Framework SDK

2.2. LOS SERVICIOS WEB

2.2.1. Introducción

A mediado de la década de los 90 y con la aparición de Internet y su posterior masificación a niveles jamás pensados, ha existido siempre la necesidad e inquietud por parte de las empresas desarrolladoras de software de buscar o contar con la manera de lograr la integración entre sistemas heterogéneos, cuando hablo de sistemas heterogéneos me refiero tanto al software como al hardware. Para tal efecto muchas compañías fueron creando de forma individual la mejor manera de lograr esta integración. Muchas empresas comenzaron

una loca carrera para generar la mejor tecnología integradora de sistemas, pero a medida que la competencia se hacia cada vez más fuerte, la integración se hacia cada vez más difícil.

Debido a la gran masificación de Internet a niveles insospechables y al gran impacto causado por las tecnologías de la información en las ultimas dos décadas del siglo pasado, la manera de hacer negocios y la comunicación entre las personas y las empresas cambió de una manera rotunda. Bajo este contexto se hacía cada vez mayor la necesidad de integrar y compartir información entre distintas plataformas de software y hardware.

Las empresas se percataron que era imposible crear una plataforma integrado de forma individual, así que decidieron atacar el problema de raíz. Para esto decidieron que en vez de crear la mejor plataforma integradora, era mejor buscar un leguaje común de intercambio de información aprovechando los estándares existentes en el mercado.

Bajo este contexto nacen los Servicios Web basados en XML, los cuales son el objeto de estudio del presente capítulo.

2.2.1.1. Objetivos

Los servicios Web nacieron con los siguientes objetivos:

- Conocer y entender el significado y alcance de los Servicios Web XML.
  Esto implica entender el contexto global en el cual se desarrollaron los Servicios Web para así conseguir de manera práctica la adopción e implementación de dicha tecnología. También conocer sus principales ventajas así como sus limitaciones desde un punto de vista de una tecnología que esta en continuo desarrollo.

- Dimensionar los nuevos cambios de paradigmas informáticos producto de la implementación de Servicios Web. Esto quiere decir, poder dimensionar que esta tecnología viene a cambiar la forma en que se comunicaban las distintas aplicaciones y la forma en que se accede a la información que reside en distintas plataformas y aplicaciones desde diversos tipos de equipos y dispositivo de comunicación.

- Ampliar el conocimiento de todas las tecnologías asociadas a los Servicios Web.

De manera general o detallada conocer las tecnologías asociadas a Servicios Web. Ya que de alguna manera, más temprano que tarde, nos encontraremos inmersos en ellas.

- Dejar un documento escrito que sirva de base para los futuros estudiantes que se interesen en descubrir nuevas tecnología.

La intención del presente documento no es solo la de explicar el contexto de los Servicios Web, si no también transmitir al lector un numero de nuevos conceptos y tecnologías que en la actualidad están en pleno desarrollo y en adopción por diversas empresas. Todo lo anterior pretende incentivar al lector a que investigue y descubra nuevas herramientas con las cuales logre estar un grado por encima del resto en lo que a informática se refiere.

2.2.1.2. Visión general

Las aplicaciones Web actuales ya no son suficientes. El modelo actual de negocio electrónico no facilita la integración de las aplicaciones de Internet con el resto de software de las empresas. Si las compañías quieren extraer el máximo beneficio de Internet, los sitios Web deben evolucionar. Este es el contexto en el que surgen los *servicios Web*. Los *servicios Web* son componentes software que permiten a los usuarios usar aplicaciones de negocio que comparten datos con otros programas modulares, vía Internet. Son aplicaciones independientes de la plataforma que pueden ser fácilmente publicadas, localizadas e invocadas mediante protocolos web estándar, como XML [13], SOAP[10], UDDI [12] o WSDL[1]. El objetivo final es la creación de un directorio online de servicios Web, que pueda ser localizado de un modo sencillo y que tenga una alta fiabilidad.

La funcionalidad de los protocolos empleados es la siguiente:

- XML (*eXtensible Markup Language*): Un servicio web es una aplicación web creada en XML.

- WSDL (*Web Services Definition Service*): Este protocolo se encarga de describir el *web service* cuando es publicado. Es el lenguaje XML que los proveedores emplean para describir sus *servicios Web*.
- SOAP (*Simple Object Access Protocol*): Permite que programas que corren en diferentes sistemas operativos se comuniquen. La comunicación entre las diferentes entidades se realiza mediante mensajes que son rutados en un sobre SOAP.
- UDDI (*Universal Description Discovery and Integration*): Este protocolo permite la publicación y localización de los servicios. Los directorios UDDI actúan como una guía telefónica de *servicios Web*.

Aunque la idea de la programación modular no es nueva, el éxito de esta tecnología reside en que se basa en estándares conocidos en los que ya se tiene una gran confianza, como el XML. Además, el uso de los *servicios Web* aporta ventajas significativas a las empresas. El principal objetivo que se logra, es la interoperabilidad y la integración. Mediante los *servicios Web*, las empresas pueden compartir servicios software con sus clientes y sus socios de negocio. Esto ayudará a las compañías a escalar sus negocios, reduciendo el coste en desarrollo y mantenimiento de software, y sacando los productos al mercado con mayor rapidez. La integración de aplicaciones hará posible obtener la información demandada en tiempo real, acelerando el proceso de toma de decisiones. La evolución de Internet hacia los *servicios Web*, mejorará los resultados globales de las empresas, reduciendo sus gastos y guiándolas hacia una mejora progresiva de la calidad. La adopción de la tecnología de servicios web por la industria es el primer paso hacia una economía global.

2.2.1.3. Posibles riesgos

Las expectaciones alrededor de esta tecnología son grandes, porque el mercado de aplicación es muy amplio. Pero también tiene sus puntos oscuros:

- Los *servicios Web* hacen uso de las mismas tecnologías que han sido atacadas en tantas ocasiones. Usando *servicios Web*, la seguridad de una empresa puede verse comprometida. La ausencia de técnicas de seguridad estándar es un obstáculo para la adopción de la tecnología.
- La calidad de un servicio Web es un parámetro que no queda demasiado claro, pero cuya medida es fundamental a la hora de desarrollar un servicio maduro.

2.2.1.4. Seguridad

Actualmente, los *servicios Web* están siendo ampliamente aceptados por las empresas para el desarrollo de software de uso interno. De este modo, los servicios pueden implementar toda su funcionalidad y permanecer seguros tras el Cortafuegos de la compañía. Los desarrollos actuales no ayudan a la cooperación entre las empresas ya que no hay ningún estándar establecido sobre las técnicas de seguridad. Debido a la tecnología que es usada por los *servicios Web*, y en concreto al uso de SOAP, las técnicas de seguridad convencionales que se han venido usando en Internet, ya no son suficientes. Con SOAP, cada mensaje simple que se intercambia realiza múltiples saltos y es rutado a través de numerosos puntos antes de que alcance su destino final. Es por ello que los *servicios Web* necesitan tecnologías que protejan los mensajes desde el principio hasta el final. Existen un conjunto de técnicas que se pueden usar para garantizar la seguridad a nivel de mensaje. Estas son:

- Cifrado XML: Evita que los datos se vean expuestos a lo largo de su recorrido.
- Firma Digital XML: Asocia los datos del mensaje al usuario que emite la firma, de modo que este usuario es el único que puede modificar dichos datos.
- XKMS y los Certificados: XKMS (*XML Key Management Specification*) define servicios Web que se pueden usar para chequear la confianza de un certificado de usuario.
- SAML y la Autorización: SAML (Security Assertion Mark-up Language) hace posible que los servicios Web intercambien información de autentificación y autorización entre ellos, de modo que un *Web Service* confíe en un usuario autentificado por otro *Web Service*.
- Validación de datos: Permite que los servicios Web reciban datos dentro de los rangos esperados.

Además, también hay técnicas que permiten mantener la seguridad a otros niveles. La seguridad en UDDI permite autentificar todas las entidades que toman parte en la publicación de un *Web Service*: proveedor, agente y consumidor del servicio. De este modo, nadie podrá registrar servicios en el papel de un proveedor o hacer uso de ellos sin contar con los permisos adecuados.

2.2.1.5. Calidad

Actualmente ya existen en el mercado algunas herramientas específicamente diseñadas para medir la calidad de los *servicios Web*, pero sigue siendo necesaria una estandarización sobre este tema. Los resultados sobre la calidad de diferentes *servicios Web*, servirán como parámetro de comparación y ayudarán al consumidor a decantarse por un servicio u otro. Para que un *Web Service* se ejecute con corrección y satisfaga las expectativas creadas, a parte del precio, habrá que tener en cuenta una serie de parámetros como por ejemplo, que los resultados obtenidos del mismo sean los esperados o que el entorno de uso sea amigable. Otro elemento a tener en cuenta es la integración. Aunque teóricamente los *servicios Web* proporcionan conectividad con cualquier software de un modo transparente, cada proveedor de servicios puede adoptar soluciones diferentes que resultan más o menos adecuadas para el consumidor. Analizando la escalabilidad se comprobará el grado de modularidad y flexibilidad del servicio. Por último, también sería interesante analizar las características que ofrece el proveedor de *servicios Web*. Actualmente no hay definidos estándares sobre este tema, pero la mayoría de las empresas ya está demandando algún tipo de acuerdo o contrato con los proveedores, de modo que se pueda garantizar la calidad y la fiabilidad de los servicios por los que se paga.

2.2.2. XML Web Services

Antes de continuar y con el propósito de dejar al lector con una idea lo más clara posible acerca del concepto de *Web Services* (Servicio Web), quiero citar una definición que rescate al asistir a una charla técnica de XML *Web Services* en Microsoft en octubre del 2003 cuyo expositor fue el señor Marcos Escovar.

Un *Web Service* es un componente de software que se comunica con otras aplicaciones codificando los mensaje en XML y enviando estos mensaje a través de protocolos estándares de Internet tales como el *Hypertext Transfer Protocol* (HTTP). Intuitivamente un *Web Service* es similar a un sitio web que no cuenta con un interfaz de usuario y que da servicio a las aplicaciones en vez de a las personas. Un *Web Service*, en vez

de obtener solicitudes desde el navegador y retornar páginas web como respuesta, lo que hace es recibir solicitudes a través de un mensaje formateado en XML desde una aplicación, realiza una tarea y devuelve un mensaje de respuesta también formateado en XML .

Microsoft y otras empresas líderes están promocionando SOAP como estándar de los mensajes para los Servicios Web. Un mensaje SOAP se parece mucho a una carta: es un sobre que contiene una cabecera con la dirección del receptor del mensaje, un conjunto de opciones de entrega (tal como la información de encriptación), y un cuerpo o body con la información o data del mensaje.

Microsoft y otros proveedores líderes promocionan los Servicios Web como un modelo de programación para la comunicación entre aplicaciones. Estas compañías piensan que la conexión de aplicaciones a través de Internet mejorará la capacidad de las empresas para trabajar conjuntamente con sus socios de negocio, proveedores y clientes. Creando una capa de servicio Web sobre una aplicación corporativa existente, las organizaciones podrán permitir que sistemas externos puedan invocar las funciones de la aplicación a través de Internet (o una intranet corporativa) sin tener que modificar la aplicación misma. Por ejemplo, varias compañías están hoy en día creando servicios Web que actúan como *front end* para aplicaciones de entrada de órdenes que están residentes internamente en un *mainframe*. Estas compañías permiten a los sistemas de compras de sus clientes enviar órdenes de compra a través de la Internet. Poner una capa de servicios Web sobre las aplicaciones existentes es una solución muy interesante para integrar las aplicaciones desarrolladas por los diferentes departamentos y así reducir los costos de integración."

Ahora que ya tenemos una breve noción de lo que es un Servicios Web nos introduciremos en aspectos un poco más técnicos.

2.2.2.1. Requisitos de un Servicio Web

- Interoperabilidad: Un servicio remoto debe permitir su utilización por clientes de otras plataformas.
- Amigabilidad con Internet: La solución debe poder funcionar para soportar clientes que accedan a los servicios remotos desde internet.

- Interfaces fuertemente tipadas: No debería haber ambigüedad acerca del tipo de dato enviado y recibido desde un servicio remoto. Más aún, los tipos de datos definidos en el servicio remoto deben poderse corresponder razonablemente bien con los tipos de datos de la mayoría de los lenguaje de programación procedimentales.
- Posibilidad de aprovechar los estándares de Internet existentes: La implementación del servicio remoto debería aprovechar estándares de Internet existentes tanto como sea posible y evitar reinventar soluciones a problema que ya se han resuelto. Una solución construida sobre un estándar de Internet ampliamente adoptado puede aprovechar conjuntos de herramientas y productos existentes creados para dicha tecnología.
- Soporte para cualquier lenguaje: La solución no debería ligarse a un lenguaje de programación particular Java RMI, por ejemplo, esta ligada completamente a lenguaje Java. Sería muy difícil invocar funcionalidad de un objeto Java remoto desde Visual Basic o PERL. Un cliente debería ser capaz de implementar un nuevo servicio Web existente independientemente del lenguaje de programación en el que se halla escrito el cliente
- Soporte para cualquier infraestructura de componente distribuida: La solución no debe estar fuertemente ligada a una infraestructura de componentes en particular. De hecho, no se bebería requerir el comprar, instalar o mantener una infraestructura de objetos distribuidos, solo construir un nuevo servicio remoto utilizar un servicio existente. Los protocolos subyacentes deberían proporcionar un nivel base de comunicación entre infraestructura de objeto distribuidos existentes tales como DCOM y CORBA.

2.2.2.2. Bloques constructivos de Servicios Web

En el siguiente grafico se muestran los bloques constructivos principales necesarios para facilitar las comunicaciones remotas entre aflicciones.

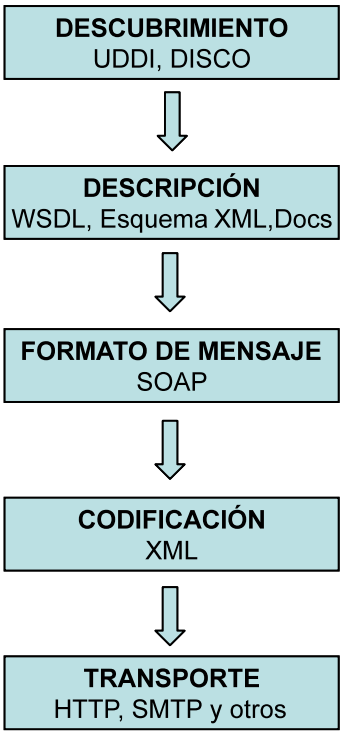


Ilustración 4: Bloques constructivos de Servicios Web

- **Descubrimiento**: La aplicación cliente que necesita acceder a la funcionalidad que expone un Servicio Web necesita una forma de resolver la ubicación de servicio remoto. Se logra mediante un proceso llamado, normalmente descubrimiento (*discovery*). El descubrimiento se puede proporcionar mediante un directorio centralizado así como por otros métodos *ad hoc*. En DCOM, el servicio de descubrimiento lo proporciona el Administrador de control de servicios (SCM, *Services Control Manager*).

- **Descripción**: Una vez que se ha resuelto el extremo de un servicio Web dado, el cliente necesita suficiente información para interactuar adecuadamente con el mismo. La descripción de un servicio Web implica meta datos estructurados sobre la interfaz que intenta utilizar la aplicación cliente así como documentación escrita sobré el servicio Web incluyendo ejemplo de uso. Un componente DCOM expone meta datos estructurados sobre sus interfaces mediante una biblioteca de tipo (*typelib*). Los meta datos dentro de una *typelib* de componente se guardan en un formato binario propietario a los que se accede mediante una interfaz de programación de aplicación (API) propietaria.

- **Formato del mensaje**: Para el intercambio de datos, el cliente y el servidor tienen que estar de acuerdo en un mecanismo común de codificación y formato de mensaje.El uso de un mecanismo estándar de codificar los datos asegura que los datos que codifica el cliente los interpretará correctamente el servidor. En DCOM los mensajes que se envían entre un cliente y un servidor tienen un formato definido por el protocolo *DCOM Object RPC* (ORPC).

- **Codificación**: Los datos que se trasmiten entre el cliente y el servidor necesitan codificarse en un cuerpo de mensaje. Dcom utiliza un esquema de codificación binaria para serializar los datos de los parámetros que se intercambian entre el cliente y el servidor.

- **Transporte**: Una vez se ha dado formato al mensaje y se han serializado los datos en el cuerpo del mensaje se debe transferir entre el cliente y el servidor utilizando algún protocolo de transporte. DCOM dispone de varios protocolos propietarios como TCP, SPX, NetBEUI y NetBIOS sobre IPX.

2.2.3. SOAP (Simple Object Access Protocol)

2.2.3.1. Descripción

Son las siglas de *Simple Object Access Protocol*. Este protocolo deriva de un protocolo creado por David Winer, XML-RPC en 1998. En su sitio Web, Userland, *http://www.userland.com/* se puede encontrar multitud de documentación acerca de este primer protocolo de comunicación bajo HTTP mediante XML. Con este protocolo se pedían realizar RPC o *remote procedure calls*, es decir, podíamos bien en cliente o servidor realizar peticiones mediante HTTP a un servidor Web. Los mensajes debían tener un formato determinado empleando XML para encapsular los parámetros de la petición. Con el paso del tiempo el proyecto iniciado por David Winer interesó a Importantes multinacionales entre las que se encuentran IBM y Microsoft y de este interés por XML-RPC se desarrollo SOAP.

En el núcleo de los servicios Web se encuentra el protocolo simple de acceso a datos SOAP, que proporciona un mecanismo estándar de empaquetar mensajes. SOAP ha recibido

gran atención debido a que facilita una comunicación del estilo RPC entre un cliente y un servidor remoto. Pero existen multitud de protocolos creados para facilitar la comunicación entre aplicaciones, incluyendo RPC de Sun, DCE de Microsoft, RMI de Java y ORPC de CORBA. ¿Por qué se presta tanta atención a SOAP.

Una de las razones principales es que SOAP ha recibido un increíble apoyo por parte de la industria. SOAP es el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, SUN, Microsystems, SAP y Ariba.

Algunas de las Ventajas de SOAP son:

- No está asociado con ningún lenguaje: los desarrolladores involucrados en nuevos proyectos pueden elegir desarrollar con el último y mejor lenguaje de programación que exista pero los desarrolladores responsables de mantener antiguas aflicciones heredadas podrían no poder hacer esta elección sobre el lenguaje de programación que utilizan. SOAP no especifica una API, por lo que la implementación de la API se deja al lenguaje de programación, como en Java, y la plataforma como Microsoft .Net.
- No se encuentra fuertemente asociado a ningún protocolo de transporte: La especificación de SOAP no describe como se deberían asociar los mensajes de SOAP con HTTP. Un mensaje de SOAP no es más que un documento XML, por lo que puede transportarse utilizando cualquier protocolo capaz de transmitir texto.
- No está atado a ninguna infraestructura de objeto distribuido La mayoría de los sistemas de objetos distribuidos se pueden extender, y ya lo están alguno de ellos para que admitan SOAP.
- Aprovecha los estándares existentes en la industria: Los principales contribuyentes a la especificación SOAP evitaron, intencionadamente, reinventar las cosas. Optaron por extender los estándares existentes para que coincidieran con sus necesidades. Por ejemplo, SOAP aprovecha XML para la codificación de los mensajes, en lugar de utilizar su propio sistema de tipo que ya están definidas en la especificación esquema de XML . Y como ya se ha mencionado SOAP no define un medio de trasporte de los mensajes; los mensajes de SOAP se pueden asociar a los protocolos de transporte existentes como HTTP y SMTP.

Permite la interoperabilidad entre múltiples entornos: SOAP se desarrollo sobre los estándares existentes de la industria, por lo que las aplicaciones que se ejecuten en plataformas con dicho estándares pueden comunicarse mediante mensaje SOAP con aplicaciones que se ejecuten en otras plataformas. Por ejemplo, una aplicación de escritorio que se ejecute en una PC puede comunicarse con una aplicación del back-end ejecutándose en un mainframe capaz de enviar y recibir XML sobre HTTP.

2.2.3.2. Anatomía de un mensaje SOAP

SOAP proporciona un mecanismo estándar de empaquetar un mensaje. Un mensaje SOAP se compone de un sobre que contiene el cuerpo del mensaje y cualquier información de cabecera que se utiliza para describir le mensaje. A continuación tiene un ejemplo:

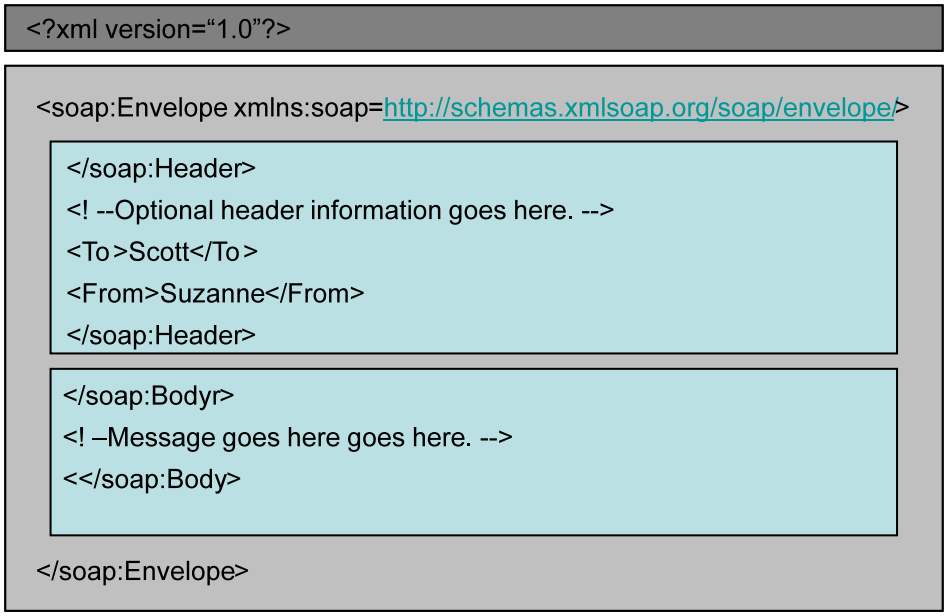


Ilustración 5: Anatomía de un mensaje SOAP

El elemento raíz del documento es el elemento *Envelope*. El ejemplo anterior contiene dos subelementos, Body y Header. Un ejemplo de SOAP valido también puede contener otros elementos hijo en el sobre. El sobre puede contener un elemento Header opcional que contiene información sobre el mensaje. En el ejemplo anterior, la cabecera contiene dos elementos que describen a quien compuso el mensaje, y posible receptor del mismo. El sobre

debe contener un elemento body el elemento body (cuerpo) contiene la carga de datos del mensaje. En el ejemplo el cuerpo contiene una simple cadena de caracteres.

Un mensaje debe estar dentro de un sobre de SOAP bien construido. Un sobre se compone de un único elemento *Envelope* el sobre puede contener un elemento Header y puede contener un elemento body. Si existe, la cabecera debe ser el elemento hijo inmediato del sobre, con el cuerpo siguiendo inmediatamente a la cabecera. El cuerpo contiene la carga de datos del mensaje y la cabecera contiene los datos adicionales que no pertenecen necesariamente al cuerpo del mensaje. Además de definir un sobre de SOAP, la especificación de SOAP define una forma de codificar los datos contenidos en un mensaje. La codificación de SOAP proporciona un mecanismo estándar para serializar tipos de datos no definidos en la parte 1 de la especificación del esquema de XML .

La especificación de SOAP también proporciona un patrón de mensaje estándar para facilitar el comportamiento de tipo RPC. Se emparejan dos mensajes de SOAP para facilitar la asociación de un mensaje de petición con un mensaje de respuesta. La llamada a un método y sus parámetros se serializan en el cuerpo del mensaje de petición en forma de una estructura. El elemento raíz tiene el mismo nombre que el método objetivo, con cada uno de los parámetros codificado como un subelemento.

El mensaje de respuesta puede contener los resultados de la llamada al método o una estructura de fallo bien definida. Los resultados de la llamada a un método se serializan en el cuerpo de la petición como una estructura. Por convenio, el elemento raíz tiene el mismo nombre que el método original al que se añade result. Los parámetros de retorno se serializan como elementos hijo, con el parámetro de retorno en primer lugar. Si se encuentra un error el cuerpo del mensaje de respuesta contendrá una estructura de fallo bien definida.

Por convenio, el elemento raíz tiene el mismo nombre que el método original al que se añade result. Los parámetros de retorno se serializan como elementos hijo, con el parámetro de retorno en primer lugar. Si se encuentra un error el cuerpo del mensaje de respuesta contendrá una estructura de fallo bien definida.

2.3. MICROSOFT *SILVERLIGHT* 2.0

2.3.1. Introducción a Silverlight

Silverlight [3] representa el siguiente paso en la experiencia de usuario mediante la tecnología Web. El objetivo de *Silverlight* es ofrecer la misma calidad de las interfaces de usuario presentes en las aplicaciones de escritorio a las aplicaciones Web, permitiendo a los desarrolladores y diseñadores Web la creación de aplicaciones acordes a las necesidades específicas de sus clientes. Ha sido diseñada para cubrir el hueco tecnológico existente entre los diseñadores y los desarrolladores ofreciéndoles un formato común con el que trabajar. Este formato será analizado y presentado por el navegador y está basado en XML, lo que hace más sencilla la creación automática de plantillas y su generación. El formato es XAML (*Extensible Applicattion Markup Language*, Lenguaje extensible de marcado de aplicaciones).

Antes de XAML, el diseñador Web utilizaba un conjunto de herramientas para expresar un diseño utilizando una tecnología que le era familiar.

El desarrollador tras ello tenía que tomar lo ofrecido por el diseñador e interpretarlo utilizando su tecnología. El diseño no se transfería necesariamente bien a la diferente tecnología de desarrollo y el desarrollador debía realizar modificaciones que podían alterar el diseño. Con *Silverlight*, el diseñador puede utilizar herramientas para expresar un diseño con XAML y el desarrollador puede tomar ese XAML, activarlo con código y desplegarlo.

Microsoft *Silverlight* es un complemento multiplataforma desarrollado para ofrecer experiencias multimedia y aplicaciones interactivas a través de la Web. Dispone de un completo modelo de programación que soporta *AJAX, .NET* y lenguajes dinámicos como *Python* y *Ruby*. *Silverlight 1.0* es programable empleando las tecnologías Web actuales, como *AJAX*, *JavaScript* y *DHTML*. *Silverlight* 2 añade soporte de lenguajes dinámicos y .NET, así como un conjunto de nuevas características que sólo son posibles empleando .NET Framework, como el almacenamiento aislado, conexión de red y conjuntos de controles avanzados.

2.3.2. El Usuario y Silverlight

Concentrándonos en la Web, la Ilustración 6: Modelos de programación y presentación Web, muestra los modelos de presentación y programación disponibles hoy en día.

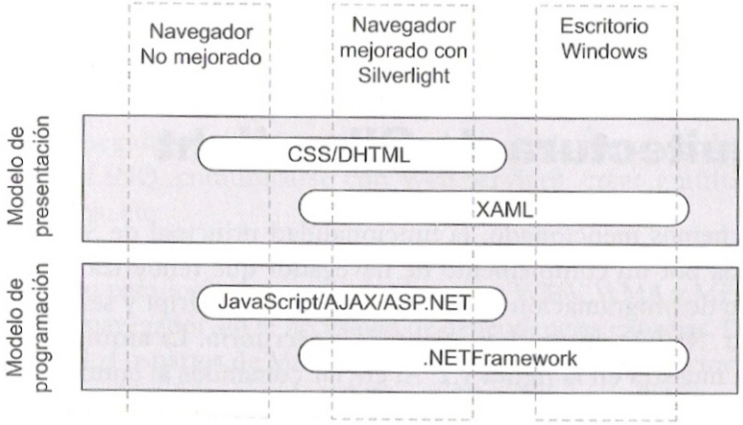


Ilustración 6: Modelos de programación y presentación Web

Como puede verse, las tecnologías típicas de desarrollo basado en navegadores disponen de *CSS/DHTML* en el modelo de presentación y *JavaScript/AJAX* en el modelo de desarrollo. En los desarrollos de escritorio, con .NET Framework 3.x, *XAML* implementa el modelo de presentación y el propio Framework el modelo de desarrollo. *Silverlight* solapa los dos modelos.

La típica aplicación interactiva está basada en tecnologías que existen en los navegadores sin *Silverlight*. La típica aplicación de escritorio se encuentra en el otro extremo del espectro, y utiliza tecnologías no relacionadas. La oportunidad de unir ambas en una aplicación y que se ejecute en el navegador se aprovecha en los navegadores con *Silverlight* que ofrecen el modelo de diseño mediante CSS/DHTML y XAML y el modelo de programación mediante *JavaScript/AJAX/.NET Framework*.

Silverlight logra esto mediante un complemento de navegador que mejora la funcionalidad del mismo añadiéndole las tecnologías típicas que proporcionan ricas interfaces de usuario, como animaciones basadas en línea de tiempos, gráficos vectoriales y contenidos

audiovisuales. Todo ello gracias al motor de presentación de *XAML*. La interfaz de usuario puede ser diseñada como *XAML* y, dado que *XAML* es un lenguaje basado en XML, y éste es texto, las aplicaciones son compatible con los cortafuegos y (potencialmente) susceptible de ser sometida a operaciones de búsqueda. El navegador XAML, lo interpreta y lo presenta.

Cuando todo esto se combina con tecnologías como *AJAX* y *JavaScript* puede convertirse en un proceso dinámico: es posible descargar pequeños trozos *XAML* y tras ello añadirlos a su interfaz gráfica, o puede editar, reordenar o eliminar *XAML* que se encuentre en el árbol de presentación utilizando simplemente programación en *JavaScript*.

2.3.3. La arquitectura de Silverlight

Como hemos comentado, la funcionalidad principal de *Silverlight* es implementada por un complemento de navegador que renderiza XAML y ofrece un modelo de programación que puede emplear *JavaScript* y ser basado en navegadores o .NET Framework y orientado a escritorio. La arquitectura que soporta esto se muestra en la Ilustración 7: La arquitectura *Silverlight* siguiente. Al enviar comandos al control en el navegador, la interfaz principal de programación que se expuso en *Silverlight* 1.0 es API de *JavaScript* DOM. Esto nos permite capturar eventos de usuarios que son generados en el interior de la aplicación (como movimientos de ratón o clics sobre elementos determinados) y disponer de código que se ejecute como respuesta a los mismos. Podemos llamar a métodos de *JavaScript* DOM sobre los elementos XAML para manipularlos permitiendo, por ejemplo, controlar la reproducción multimedia o el inicio de animaciones.

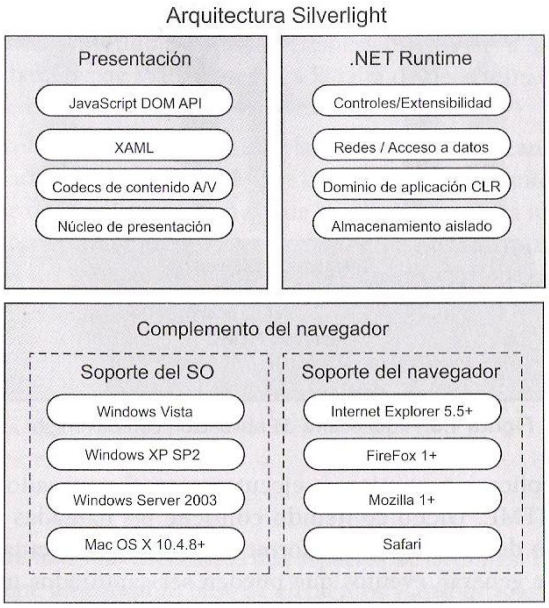


Ilustración 7: La arquitectura *Silverlight*

También podemos programar una aplicación renderizada por el control utilizando el entorno común de ejecución de lenguajes (CLR) de .NET Framework. Además de lo que podemos hacer con *JavaScript*, esto nos permite utilizar gran parte de los espacios de nombre y controles incluidos en .NET Framework, logrando resultados que son muy difíciles de conseguir, si no imposible, con *JavaScript*, como acceder a datos con ADO.NET y LINQ, comunicarse con Web Services, crear y utilizar controles personalizados, etc.

Adicionalmente, el entorno de ejecución de la presentación incluye todo el software necesario para lograr que tecnologías como WMV, WMA, MP3 sean reproducidas en el navegador sin la necesidad de dependencias externas. De este modo, por ejemplo los usuarios de Macintosh no necesitan el reproductor de Windows Media para reproducir contenido WMV, con *Silverlight* es suficiente.

La arquitectura de una aplicación sencilla ejecutándose en el navegador utilizando *Silverlight* se muestra a continuación:

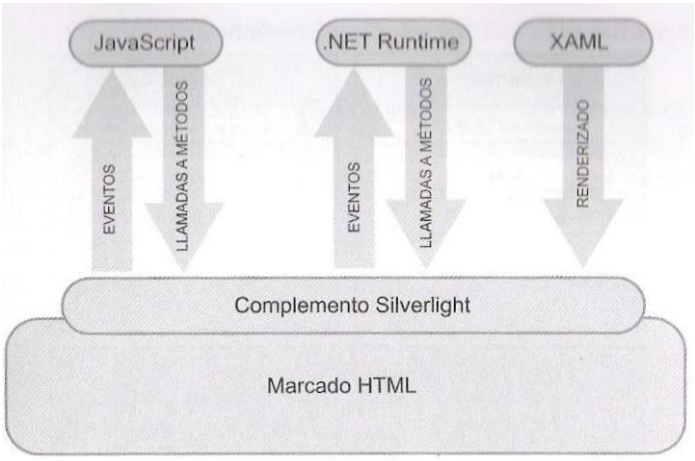


Ilustración 8: Arquitectura de aplicación con *Silverlight*

Cuando la aplicación empieza a ejecutarse en el navegador, se interpreta su contenido HTML. Dicho contenido contiene las llamadas para instanciar el complemento de *Silverlight*. Conforme el usuario interactúa con la aplicación *Silverlight*, se generan eventos que pueden ser capturados mediante funciones *JavaScript* o .NET Framework. Por su parte, el código del programa puede realizar llamadas a métodos que actúan sobre los elementos del contenido *Silverlight* para manipularlo, añadir nuevo contenido o eliminar el existente. El contenido XAML puede encontrarse en la propia página, externamente en un archivo estático o ser XAML dinámico generado por un servidor.

2.3.4. XAML

La tecnología base que logra toda esta experiencia descrita anteriormente es XAML, éste es un lenguaje basado en XML empleado para definir los elementos visuales de su aplicación. Ello incluye interfaces de usuario, elementos gráficos, animaciones, contenido multimedia, controles y mucho más. Fue introducido por Microsoft para Windows Presentation Foundation (WPF), conocida anteriormente como Avalon, que es una tecnología orientada al escritorio parte de .NET Framework 3.0 y posteriores. Ha sido diseñada, como se ha indicado anteriormente, para tender un puente entre diseñadores y desarrolladores durante la creación de aplicaciones.

2.3.5. Servicios Web en Silverlight

2.3.5.1. Acceso a Servicios Web en Silverlight

Las aplicaciones cliente de *Silverlight* se ejecutan en el explorador y, a menudo, necesitan tener acceso a los datos desde varios orígenes externos. Un ejemplo típico es el acceso a los datos desde una base de datos en un servidor *back-end* y la visualización de los mismos en la interfaz de usuario de *Silverlight*. Otro escenario común es la actualización de datos en un servicio *back-end* a través de la interfaz de usuario de *Silverlight* que realiza envíos a ese servicio. A menudo, dichos orígenes de datos externos adquieren la forma de servicios web.

Éstos pueden ser servicios SOAP creados mediante *Windows Communication Foundation (WCF)* o cualquier otra tecnología SOAP. También pueden ser servicios HTTP o REST sin formato. Los clientes de *Silverlight* pueden tener acceso a estos servicios Web directamente o, en el caso de los servicios SOAP, mediante un proxy generado a partir de metadatos que haya publicado el servicio en cuestión.

Silverlight también ofrece la funcionalidad necesaria para trabajar con los distintos formatos de datos que usan los servicios. Los formatos pueden ser *XML, JSON, RSS y Atom*. Es posible tener acceso a los formatos de datos mediante los componentes de serialización, *Linq to XML*, *Linq to JSON* y *Syndication*. Los servicios web a los que una aplicación de *Silverlight* puede tener acceso deben cumplir con determinadas reglas a fin de permitir dicho acceso.

2.3.5.2. Acceso a un servicio web en Silverlight con Visual Studio

En este tema se describe cómo tener acceso a un servicio web [4] desde un cliente de *Silverlight* mediante un proxy. Un proxy es una clase que le ayuda a tener acceso a un servicio determinado; se crea automáticamente mediante la herramienta *"Agregar referencia de servicio"* en Visual Studio 2008. Debe crear un proxy independiente para cada servicio al que desee tener acceso. Se puede utilizar el mismo procedimiento que se describe aquí para tener acceso a un servicio SOAP público en caso de que se conozca la dirección del servicio en cuestión y siempre que dicho servicio permita tener acceso mediante aplicaciones del explorador desde el dominio de la aplicación.

Para agregar una referencia de servicio al servicio:

1. Haga clic con el botón secundario en el nombre del proyecto en el *Explorador de soluciones* y, a continuación, seleccione *Agregar referencia de servicio*.
2. Haga clic en el botón *Descubrir* para localizar el Servicio. En caso de que el servicio sea externo, previamente se deberá introducir la URL del servicio.
3. Introduzca el nombre del espacio de nombres que recibirá el Servicio y a continuación haga clic en Aceptar.
4. Se observará que en el Explorador de soluciones se ha agregado una carpeta llamada Referencias de Servicio. Se pueden examinar los miembros de dicho Servicio haciendo clic en el botón derecho y seleccionando *Vista en el explorador de objetos*.

Realizar llamadas a operaciones en el Servicio Web

1. Todas las llamadas del servicio web de *Silverlight [5]* son asincrónicas. El proxy contiene dos miembros para cada operación del servicio: un método asincrónico y un evento completo. Considerando la operación del servicio `ExecJob`. En primer lugar, se agrega `EventHandler` a `ExecJobCompleted`: este evento se invocará cuando el servicio devuelva los datos que solicitamos. Después de la configuración del evento, podemos realizar una llamada al servicio llamando a `ExecJobAsync`.
2. `proxy. ExecJobCompleted += new EventHandler<CountUsersCompletedEventArgs>(proxy_ExecJobCompleted);`
3. `proxy. ExecJobAsync();`
4. El controlador de eventos especifica que se debe llamar al método `proxy_ExecJobCompleted` cuando el servicio devuelva algunos datos. Necesitamos definir y agregar este método en la Page.
5. `void proxy_ExecJobCompleted(object sender, ExecJobCompletedEventArgs e)`
6. `{ ExecJobResult.Text = "Job: " + e.Result; }`

Condiciones de error

Se pueden producir varios errores al llamar a los servicios. Por ejemplo, puede que el servicio no se encuentre disponible, que no esté configurado para admitir protocolos que

Silverlight entienda o que la dirección del servicio no sea correcta. También se pueden producir errores derivados de problemas de acceso entre dominios. El servicio al que intenta conectarse puede no publicar una directiva entre dominios adecuada y, por tanto, no se permitirá el acceso desde las aplicaciones basadas en el explorador.

En otra clase específica de errores, puede encontrarse un error concreto aplicable a un servicio determinado que se envía como un error de SOAP. Un ejemplo de ello puede ser que se haya intentado llamar a `ExecJob` con un parámetro incorrecto.

Configuración de la dirección del servicio

Para las aplicaciones de *Silverlight*, el archivo de configuración del cliente contiene un subconjunto de configuración del cliente de Windows Communicaction Foundation (WCF). El nombre del archivo de configuración de *Silverlight* debe ser `ServiceReferences.Clientconfig` y también se debe empaquetar e implementar junto con la aplicación.

La herramienta Agregar referencia de servicio genera automáticamente la configuración del cliente que se encuentra en el archivo `ServiceReferences.ClientConfig`.

La configuración se compone de una sección de `<bindings>` y de una sección de `<client>`, como se puede observar en el ejemplo siguiente. Las secciones se encuentran en el elemento `<system.serviceModel>`. Los elementos que se encargan de configurar el cliente de `server_qsub` para tener acceso al servicio web de Service1, que cuenta con un contrato de `IService`, se encuentran en el elemento `<system.serviceModel>` que, a su vez, se encuentra en el elemento `<configuration>`.

```
<configuration>
    <system.serviceModel>
        <bindings>
            <basicHttpBinding>
                <binding name="server_qsubSoapBinding"
maxBufferSize="2147483647"
                    maxReceivedMessageSize="2147483647">
                    <security mode="None" />
                </binding>
```

```
            </basicHttpBinding>
        </bindings>
        <client>
            <endpoint
address="http://192.168.1.212:8080/server_qsub/services/server_qsub"
                binding="basicHttpBinding"
bindingConfiguration="server_qsubSoapBinding"
                contract="server_qsub.server_qsub" name="server_qsub" />
        </client>
    </system.serviceModel>
</configuration>
```

El atributo address del elemento `<endpoint>` se encuentren en el mismo atributo del contract.

Configuración del método de seguridad

```
<client>
   <endpoint
      address="http://uk.example.com/Service1.svc"
      binding="basicHttpBinding"
      bindingConfiguration="BasicHttpBinding_IService"
      contract="CustomerClient.CustomerService.IService"
      name="EuropeDataCenter" />
   <endpoint
      address="http://jpn.example.com/Service1.svc"
      binding="basicHttpBinding"
      bindingConfiguration="BasicHttpBinding_IService"
      contract="CustomerClient.CustomerService.IService"
      name="AsiaDataCenter" />
    </client>
```

2.3.5.3. Permisos para el acceso a Servicios Web en otro dominio

Por razones de seguridad, un servicio web no se puede consumir desde *Silverlight* sin que se especifiquen los permisos en un fichero. Este fichero puede ser `clientaccesspolocy.xml` o `crossdomain.xml` y se debe emplazar en el directorio raíz del servidor que ofrece el Servicio Web. A continuación se exponen dos ejemplos para cada uno de los ficheros:

- *Clientaccesspolicy.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
<cross-domain-access>
<policy>
<allow-from http-request-headers="*">
<domain uri="*"/>
</allow-from>
<grant-to>
<resource path="/" include-subpaths="true"/>
</grant-to>
</policy>
</cross-domain-access>
</access-policy>
```

- *Crossdomain.xml*

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

2.4. PLATAFORMA JAVA

2.4.1. Web Services en Java

Para la realización del presente proyecto, se han utilizado Servicios Web en Java, . En concreto, una implementación de código libre llamada Apache Axis. A continuación se expone una explicación de la herramienta.

2.4.2. Introducción al Apache Axis

Apache Axis es una implementación de código libre de SOAP que proporciona un entorno de ejecución para Servicios Web implementados en Java. A grandes rasgos, un Servicio Web es un conjunto de métodos que realizan una funcionalidad que se exponen al resto de las aplicaciones.

Cualquier aplicación sea cual sea su plataforma o lenguaje en la que está implementada podrá invocar los métodos que expone el Servicio Web, por ejemplo, una aplicación .Net (Implica una plataforma Windows) podría invocar métodos expuestos por un Servicio Web Java ejecutándose en una plataforma Linux. Esto se consigue utilizando protocolos estándar como XML y HTTP y se evitan los problemas con Firewalls, etc. que otras tecnologías similares como CORBA o RMI tenían.

Entre otras cosas Axis proporciona:

- Un entorno de ejecución para Servicios Web Java (*.jws)
- Herramientas para crear WSDL desde clases java.
- Herramientas para crear clientes Java desde un WSDL.
- Herramientas para desplegar, probar y monitorizar Servicios Web.
- Integración con servidores de aplicaciones y contenedores de Servlets.

2.4.3. Instalación de Axis sobre Tomcat

1. Descargar la versión binaria de Axis desde HTTP://WS.APACHE.ORG/AXIS/
2. Descomprimir el fichero.
3. Copiar el directorio completo axis al directorio: TOMCAT_HOME/webapps/axis

2.4.4. Creación de un cliente de Web Service

Ahora vamos a hacer un cliente a manita utilizando las clases de Axis.

El cliente simplemente multiplicará dos números, pero si observa el código fuente este es bastante didáctico. Observe el texto en negrita, pues estos deben ser iguales a los que contiene el WSDL. Es decir, deben coincidir la dirección del servicio, el nombre del método a invocar y el nombre de los parámetros necesarios para invocar el método.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.ParameterMode;
/**
* Prueba el Web Service
* @author Rafael Montero Montero
*/
public class Clase_ejemplo {
  public static void main(String [] args) throws Exception {
    String endpoint =
  "http://localhost:8080/axis/services/CalculadoraWS";
    Integer operando1 = new Integer(20);
    Integer operando2 = new Integer(30);
    Service service = new Service();
    Call call = (Call) service.createCall();
    // Establecemos la dirección en la que está activado el WebService
    call.setTargetEndpointAddress( new java.net.URL(endpoint) );
    // Establecemos el nombre del método a invocar
    call.setOperationName( "suma" );
    // Establecemos los parámetros que necesita el método
    // Observe que se deben especificar correctamente tanto el nómbre
  como el tipo de datos..
    // esta información se puede obtener viendo el WSDL del servicio Web
    call.addParameter( "in0", XMLType.XSD_INT, ParameterMode.IN );
    call.addParameter( "in1", XMLType.XSD_INT, ParameterMode.IN );
    // Especificamos el tipo de datos que devuelve el método.
    call.setReturnType( XMLType.XSD_INT );
    // Invocamos el método
    Integer result = (Integer) call.invoke( new Object [] { op1, op2 });
    // Imprimimos los resultados
    System.out.println("El resultado de la multiplicación es: " +
  result);
```

```
    }
}
```

2.4.5. Conclusiones

Apache Axis es una implementación sólida, madura y extendida para ejecutar, testear y administrar Servicios Web implementados en Java. Además detrás de Axis hay importantes organizaciones como Apache, por lo que eso nos da confianza a la hora de elegir Axis como entorno de ejecución de servicios Web.

2.5. TORQUE

TORQUE (Tera-scale Open-source Resource and QUEue manager**)** [8] gestiona los trabajos por lotes (*batch*). Es un producto *OpenSource*, basado en el gestor de trabajos por lotes *OpenPBS*. Ha sido desarrollado por la NASA y es utilizado por *SuperCluster consortium*, y numerosas universidades y laboratorios de investigación.

Torque es un gestor de recursos que proporciona control sobre trabajos batch y nodos de cómputo distribuidos. Utiliza un mecanismo de colas para la ejecución de trabajos que respeta los criterios de prioridad configurados.

Torque, a través de un interfaz de comandos de usuario y una librería API, permite:

- Enviar un trabajo.
- Mostrar el estado y las características de un trabajo.
- Cancelar un trabajo.
- Cambiar las características de un trabajo en espera o en ejecución (para un trabajo en ejecución, sólo puede cambiarse los límites y los ficheros de salida).
- Parar o reanudar un trabajo.
- Gestionar más de 5000 trabajos activos o en espera.

La librería API permite hacer un seguimiento de los trabajos y ejecutar el envío de trabajos. Esta API suministra la siguiente información: nombre de usuario, número de trabajo,

clase de ejecución, prioridad, recursos usados incluyendo número de procesadores, tamaño de memoria, CPU y tiempos.

Cuando el cluster (o parte de él) es usado, un mecanismo interactúa con el gestor de trabajos batch para definir si un trabajo puede ser ejecutado.

El gestor de trabajos batch distribuye los trabajos en los nodos siguiendo diferentes criterios, tales como:

- La potencia de cálculo de los nodos.
- Cómo es usada la memoria en cada nodo.
- Número de trabajos en ejecución en los nodos.
- Fechas asociadas a cada trabajo (envío, comienzo, finalización), etc.

El gestor de trabajos batch permite definir los límites para cada recurso. Si se sobrepasa el límite de un recurso en un nodo, ningún trabajo nuevo será asignado a este nodo. Asimismo, los trabajos en ejecución de este nodo serán suspendidos, reenviados o parados.

El comando *Torque* para envío de trabajos es **qsub**

Para obtener información más detallada, ver HTTP://WWW.SUPERCLUSTER.ORG/TORQUE o HTTP://HTTP://WWW.OPENPBS.ORG.

2.5.1. Monitoreo de la ejecución de trabajos con qstat

El comando qstat muestra el estado de los trabajos de PBS. La sintaxis es:

```
%qstat [opciones] Opciones
```

Las opciones de qstat más comunes son:

-a Muestra todos los procesos del usuario.

-q Despliega la información básica mas el nombre de las colas.

-s Muestra la información básica mas la suministrada por el -calendarizador. –Q Despliega información básica y el nombre de las colas.

-f Muestra detalles de la ejecución de los trabajos del usuario.

Para más información y opciones teclear:

```
%man qstat
```

La salida de la instrucción qstat es:

```
[hank@machine]$ qstat -q
Job id Name User Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
3536.xj5 myjob.tcsh hank 0 R cola1
```

La cual muestra el identificador del trabajo (*3536.xj5*).La información mostrada es el nombre del shell script (*myjob.tcsh*), el nombre de usuario (*hank*), el estado en el que se encuentra el trabajo (*Rejecutándose*) y el nombre de la cola (*cola1*) que se encuentra atendiendo este proceso.

2.5.2. Eliminar trabajos con qdel

El comando qdel elimina un trabajo ejecutándose en PBS. La sintaxis es:

```
%qdel [opciones] identificador_del_proceso
```

La instrucción qdel *3536.xj5* elimina de PBS el trabajo con identificador *3536.xj5*.

2.5.3. Creando un Shell script para trabajo secuencial

Para ejecutar un trabajo en el entorno de PBS es necesario crear un *shell script* que contenga los comandos a ejecutar y los recursos necesarios para su ejecución. Dentro de este archivo se pueden incluir:

- Ordenes y sentencias de control de PBS.
- Opciones del comando qsub.
- Instrucciones del sistema.
- Comentarios.

Un trabajo secuencial puede ejecutarse de dos maneras: en "*modo dedicado*" y en "*modo compartido*".

En modo dedicado se reserva un procesador dedicado exclusivamente para el trabajo, pero su ejecución se iniciará hasta que haya un procesador disponible. En modo compartido el programa comenzará a ejecutarse inmediatamente, pero compartirá los recursos con otros trabajos.

Lo siguiente es un ejemplo de un shell script llamado `mishell` que ejecuta un trabajo secuencial en modo compartido:

```
#*************************************************************************
#!/bin/bash
#PBS -q batch
#PBS -o myout
#PBS -e errors
#PBS -l mem=512Mb
cd /home/hank
# Compilar el programa con algunas opciones
gcc -o myprog myprog.c
# Correr el programa
/home/hank/myprog
#Borrar el ejecutable para no saturar el disco
rm myprog
#
#*************************************************************************
```

Para ejecutar este trabajo, escribimos:

```
%qsub mishell
```

La salida de la instrucción anterior es: *3536.xj5*, que nos indica el identificador del proceso. Este shell script generaría dos archivos de salida, además de los que genere el mismo programa:

- `myout`: contiene toda la salida a la que se redirigirá la salida estándar (el monitor).
- `errors`: El segundo contiene los errores de la compilación -si los hubiere- y en general los errores generados por la ejecución del shell script.

Si las opciones `#PBS -o` y `#PBS -e` no se especifican, PBS genera dos archivos llamados *myshell.o3536* y *myshell.e3536*, cuyos nombres están formados por el nombre del shell script mas 'o' (para la salida estándar) y 'e' (para los errores) más el identificador del trabajo, en este caso 3536.

Por otra parte, el siguiente ejemplo muestra como enviar un trabajo secuencial en modo dedicado.

```
#*********************************************************************
#!/bin/bash
#PBS -q batch
#PBS -o myout
#PBS -e errors
#PBS -l nodes=1
#PBS -l mem=512Mb
cd /home/hank
# Compilar el programa con algunas opciones
gcc -o myprog myprog.c
# Correr el programa
/home/hank/myprog
#Borrar el ejecutable para no saturar el disco
rm myprog
#
*********************************************************************
```

```
%qsub mishell
```

La diferencia entre ambos ejemplos es la opción `#PBS -l nodes=1` la cual especifica que solicitamos un solo nodo para este trabajo.

NOTAS

* Cuando las opciones figuran dentro del archivo, éstas deben especificarse al principio del mismo, una por cada línea y precedidas por #PBS, sin dejar espacios. Por ejemplo:

#PBS -l mem=50mb

2.5.4. Shell script para un trabajo usando MPI

Ejemplo de un shell script llamado mishell:

```
#*********************************************************************
#!/bin/bash
#PBS -q batch
#PBS -l nodes=8
#PBS -l mem=370Mb
cd /home/hank/
cat `echo $PBS_NODEFILE`
time mpirun -machinefile $PBS_NODEFILE -np 8 /home/hank/a.out
#*********************************************************************
```

Con el shell script anterior queremos ejecutar en la cola llamada batch que tiene 500MB por nodo y 8 nodos.

Para ejecutarlo en los nodos que tenga disponibles, se deberá utilizar la variable `PBS_NODEFILE` como archivo de máquinas para `mpirun`.

Take chances, make mistakes. That's how you grow. Pain nourishes your courage. You have to fail in order to practice being brave.
(Mary Tyler Moore)

3. ANÁLISIS DEL PROYECTO

3.1. DESCRIPCIÓN GENERAL

A continuación, se citarán los principales componentes del sistema y se explicará en términos generales, la funcionalidad de cada uno.

- *Servidor del S.R.G.C.:* se encargará de lanzar los trabajos utilizando la herramienta de colas *torque*. Además, será el encargado de ofrecer los distintos servicios web que el cliente utilizará para realizar las distintas funciones.
- *Torque*: Esta herramienta se encontrará instalada y configurada en el servidor. Es una herramienta que funciona bajo el sistema operativo Linux y se encarga de gestionar diferentes colas a las que se pueden asignar tareas a realizar (*jobs*), definiendo una serie de parámetros para dicha tarea.
- *Cliente del S.R.G.C.*: implementado con la tecnología *Silverlight* de Microsoft, nos proporciona una interfaz Web para la gestión de los trabajos enviados a la cola de trabajo del servidor. Nos permite definir los parámetros con los que dicha tarea se ejecutará y nos mostrará la salida y los errores que proporcione.

La aplicación muestra una interfaz web accesible desde la red, en la cual se pueden realizar diferentes acciones, éstas son las siguientes:

- *Funcionalidad Lanzar*: Se definen los parámetros con los que se ejecutará en la cola el programa seleccionado.
- *Funcionalidad Info*: Se muestra información sobre los trabajos lanzados (la salida de vuelta y los errores si los hubiera), sobre las colas de ejecución en el servidor y el estado de los nodos que forman el cluster.
- *Funcionalidad Historial*: Se muestra un historial de los trabajos ejecutados en el servidor, desde esta pestaña podremos editar los parámetros y volver a lanzar el ejecutable.

Las tareas se ejecutarán en el servidor, concretamente, en la cola seleccionada por el usuario desde la interfaz. Una vez haya terminado la ejecución del programa, se devolverá la salida y los errores al cliente, mostrando la información en la interfaz web.

3.2. DESCRIPCIÓN DETALLADA

Se desea diseñar una Sistema que gestione los trabajos que se envían a una cola de ejecución de un servidor dedicado a ello.

Este Sistema tendrá la arquitectura típica Cliente-Servidor, el cliente se desarrollará con la tecnología *Silverlight* siendo una interfaz web sencilla y accesible desde la red. El servidor se implementará en el sistema operativo Linux el cual tendrá instalado una herramienta de ejecución de trabajos en cola llamada *Torque*. Esta herramienta gestionará las tareas que se envíen a ejecutar.

A continuación se expone un gráfico de la arquitectura del sistema y una descripción detallada de cada uno de los componentes:

Ilustración 9: Arquitectura del Sistema

Cliente

Como se ha dicho anteriormente, el cliente estará formado por una aplicación web desarrollada con la tecnología *Silverlight*. A través de ésta se podrán administrar las tareas enviadas a la cola de ejecución del servidor, así como visualizar información de los distintos elementos del sistema de colas.

Los parámetros que se pueden especificar a la hora de enviar un trabajo a ejecución son los siguientes:

- **Nombre:** Nombre del trabajo que se ejecutará. No se podrá especificar el mismo nombre que otro trabajo ya lanzado.
- **Argumentos:** Argumentos del programa enviado a ejecución.
- **Cola:** Nombre de la cola a la que se envía el trabajo. Se elije de las existentes en el servidor.
- **Memoria:** Memoria máxima disponible para la ejecución del trabajo en el servidor.
- **Tiempo Máximo:** Tiempo máximo disponible para la ejecución del trabajo.
- **Nodos:** Número de nodos que se utilizarán para la ejecución del trabajo.
- **Procesadores/Nodo:** Número de procesadores de los nodos que se utilizarán para la ejecución del trabajo.
- **Modo:** Existen dos modos de ejecución, *secuencial* o *paralelo.* En el modo secuencial los trabajos se ejecutarán en un nodo, en cambio en el modo paralelo se realizará de manera distribuida y se utilizarán paradigmas de

programación de memoria distribuida (por ejemplo MPI) para su compilación y ejecución.

- **Vuelve a ejecutar si falla:** Si el trabajo falla al ser ejecutado, se vuelve a lanzar automáticamente.
- **Exportar variables de entorno:** Se importan las variables de entorno para su ejecución.
- **Ejecutable:** Se selecciona el fichero compilado para la ejecución en el servidor.

Torque

La herramienta *Torque* proporciona un sistema de colas de ejecución el cual está formado por diferentes nodos que llevan a cabo la ejecución de los programas.

Mediante los comandos que proporciona, es posible realizar una gestión completa de las colas y sus nodos.

Servidor

El servidor será el que ofrezca la funcionalidad de le ejecución de los trabajos en las colas. Éste proveerá de los servicios necesarios para la correcta comunicación con el cliente, y su correspondiente interacción con el sistema de colas.

Los servicios se ofrecerán como Web Services y éstos serán accedidos por el cliente. A continuación se exponen los métodos remotos o funcionalidades que se proveerán:

- **Update_job:** Comprueba si se han generado los ficheros de salida y error del programa, en caso afirmativo se devuelve la respuesta al cliente. Este método es llamado periódicamente por el Cliente para verificar si el fichero ha finalizado su ejecución.

- **Get_queues:** Devuelve información sobre las colas existentes en el servidor.

- **Get_nodes:** Devuelve información sobre los equipos que forman el cluster del sistema de colas *Torque*.

- **Exec_file:** Ejecuta el trabajo y devuelve al cliente un identificador asignado por *Torque*.
- **Send_params:** Escribe los parámetros de configuración en un fichero situado en el servidor.

Una vez el servidor ha recibido los parámetros y el fichero ejecutable del cliente, éste genera un fichero PBS para su ejecución en la cola. A continuación se muestra un fichero ejemplo:

```
#!/bin/bash
#PBS -l nodes=2:ppn=1
#PBS -l walltime=300,mem=512mb
#PBS -q normal
#PBS -N mi_programa
#PBS -o /home/hank/pruebas/my_programa.out
#PBS -e /home/hank/pruebas/my_programa.err
echo " Running on " `hostname`
cd $PBS_O_WORKDIR
cat $PBS_NODEFILE > nodesinfo.txt
programa parametro1
```

Un proceso habitual de ejecución de un trabajo podría ser el siguiente:

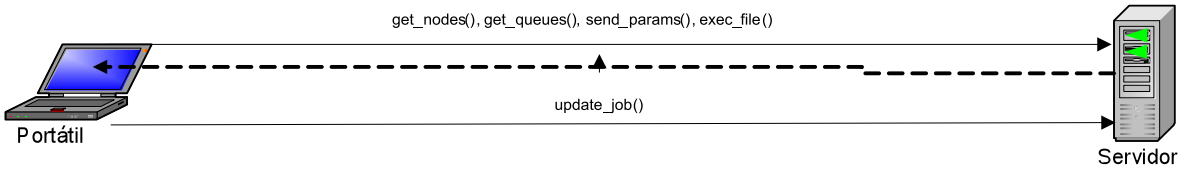


Ilustración 10: Proceso de ejecución

3.3. ESPECIFICACIÓN DE REQUISITOS DE USUARIO

El objetivo de este apartado es identificar, clasificar y catalogar los requisitos de usuario obtenidos durante las distintas sesiones de trabajo realizadas con el tutor del proyecto.

Los requisitos deberán ser validados por todos los implicados antes de llevarse a cabo la implementación del sistema. Dichos requisitos mostrarán tanto lo que el sistema debe hacer de acuerdo a lo establecido, como la manera en que lo va a hacer. Es decir, se tendrán dos tipos de requisitos diferenciados:

- **Requisitos de capacidad:** Representan aquello que el cliente desea que el sistema haga, son requisitos referidos al problema en cuestión que se quiere resolver.
- **Requisitos de restricción:** Indica la manera que tendrá el sistema de resolver los problemas, la forma de interactuar con el sistema y las restricciones que habrá sobre éste.

Para la catalogación de los requisitos se utilizarán unas tablas cuyo contenido se explica a continuación:

- **Identificador→** Cada requisito de usuario deberá estar identificado de forma unívoca con el objetivo de facilitar su trazabilidad.
- **Necesidad→**Todo requisito puede ser esencial, deseable u opcional, los requisitos esenciales se deberán cumplir a rajatabla y no son negociables. Los otros dos pueden verse sujetos a modificaciones.
- **Prioridad→**Cada requisito debe tener establecida una prioridad que sirve al desarrollador de referencia a la hora de planificar la creación del sistema.
- **Estabilidad→**Algunos requisitos pueden presentarse fijos a lo largo de toda la vida del proyecto (estables) mientras que otros dependen de otros factores y pueden estar sujetos a cambios (no estables).
- **Fuente→** Cada requisito tendrá como fuente al tutor o al alumno.

- **Descripción→** Describe de forma clara, verificable y concisa el requisito.

3.3.1. Requisitos de Capacidad

Id Requisito	**URD-C-001**
Descripción	La aplicación debe tener una interfaz gráfica. La interfaz gráfica debe permitir la administración de los trabajos enviados.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 2: URD-C-001

Id Requisito	**URD-C-002**
Descripción	Se actualizará automáticamente el estado de los nodos (Ocupado o Libre).
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 3: URD-C-002

Id Requisito	**URD-C-003**
Descripción	Se mostrará un pequeño gráfico de color verde o rojo dependiendo del estado del nodo.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 4: URD-C-003

Id Requisito	**URD-C-004**
Descripción	Se podrán indicar los parámetros del trabajo a ejecutar. Éstos serán los siguientes: Parámetros básicos ▪ Nombre del trabajo ▪ Argumentos del programa a ejecutar ▪ Cola a la que se asigna el trabajo Parámetros adicionales ▪ Memoria máxima consumida ▪ Número de nodos a utilizar ▪ Tiempo máximo en ejecución ▪ Número de procesadores por nodos utilizar ▪ Modo de ejecución: secuencial o paralelo. ▪ Re-ejecución si falla ▪ Exportar variables de entorno

Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 5: URD-C-004

Id Requisito	**URD-C-005**
Descripción	Se podrá seleccionar el fichero ejecutable (ya compilado) a ejecutar.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 6: URD-C-005

Id Requisito	**URD-C-006**
Descripción	Se podrá visualizar la siguiente información de cada nodo: ▪ Nombre del nodo ▪ Estado ▪ Número de procesadores ▪ Propiedades ▪ Tipo de nodo ▪ Estado (Sistema operativo, versión del núcleo, número de sesiones, número de usuarios, tiempo en espera, memoria total del sistema, memoria disponible en el sistema, número de cpu's del nodo, estado, trabajos)
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 7: URD-C-006

Id Requisito	**URD-C-007**
Descripción	Se podrá visualizar la siguiente información sobre los trabajos: ▪ Identificador ▪ Nombre ▪ Argumentos ▪ Qsub Id ▪ Nodos ▪ Procesadores / Nodo ▪ Cola ▪ Fecha ▪ Modo ▪ Tiempo máximo de ejecución ▪ Memoria máxima disponible ▪ Salida normal ▪ Salida de error.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 8: URD-C-007

Id Requisito	**URD-C-008**
Descripción	Se podrá visualizar la siguiente información sobre las colas: ▪ Nombre ▪ Máxima memoria disponible para el trabajo ▪ Máximo tiempo de cpu disponible para un trabajo ▪ Máximo tiempo en la cola para un trabajo ▪ Número máximo de nodos disponibles ▪ Número de trabajos en ejecución ▪ Número de trabajos en cola ▪ Número máximo de trabajos en ejecución a la vez ▪ Estado de la cola
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 9: URD-C-008

Id Requisito	**URD-C-009**
Descripción	Se guardará un historial de los trabajos ejecutados.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 10: URD-C-009

Id Requisito	**URD-C-010**
Descripción	Se podrá relanzar un trabajo realizado desde el historial, éste se ejecutará con los mismos parámetros que fue ejecutado.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 11: URD-C-010

Id Requisito	**URD-C-011**
Descripción	Se podrá editar un trabajo ejecutado con la finalidad de cambiar sus parámetros y relanzarlo.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 12: URD-C-011

Id Requisito	**URD-C-012**
Descripción	El Sistema debe permitir la conexión de varios clientes.
Necesidad	**Esencial** / No esencial
Prioridad	Alta / **Media** / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Alumno**

Tabla 13: URD-C-012

3.3.2. Requisitos de Restricción

Id Requisito	**URD-R-001**
Descripción	Para el funcionamiento de la aplicación será necesario instalar el sistema de colas *Torque* en el Sistema Operativo Linux, donde residirá el servidor.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Alumno**

Tabla 14: URD-R-001

Id Requisito	**URD-R-002**
Descripción	El módulo del servidor funcionará sobre un sistema operativo Linux.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 15: URD-R-002

Id Requisito	**URD-R-003**
Descripción	Se debe contar con uno o más equipos disponibles los cuales tendrán instalados el sistema de colas *Torque* para la ejecución de las tareas.
Necesidad	**Esencial** / No esencial
Prioridad	Alta / **Media** / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Alumno**

Tabla 16: URD-R-003

Id Requisito	**URD-R-004**
Descripción	El cliente de la aplicación será una interfaz web en *Silverlight*.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Tutor**

Tabla 17: URD-R-004

Id Requisito	**URD-R-005**
Descripción	El fichero debe estar compilado antes de ser cargado en la aplicación.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Alumno**

Tabla 18: URD-R-005

Id Requisito	**URD-R-006**
Descripción	La comunicación entre cliente (interfaz en *Silverlight*) y servidor (Linux), deberá ser mediante la tecnología *Web Services*.
Necesidad	**Esencial** / No esencial
Prioridad	**Alta** / Media / Baja
Estabilidad	Con cambios **/ Sin cambios**
Fuente	**Alumno**

Tabla 19: URD-R-006

3.3.3. Descomposición en Subsistemas

Una de las técnicas que más se utilizan en el diseño de software es separar el sistema en componentes más pequeños. De esta forma, será más fácil su diseño y su posterior implementación, así como su depuración y mantenimiento.

Tras lo descrito, se aprecia una clara arquitectura Cliente-Servidor. Así pues el cliente recibiría las tareas a realizar. El cliente se encargaría de la presentación mediante una interfaz web y sería el encargado de enviar los trabajos al servidor, y la administración de los mismos.

A continuación se describen las funcionalidades de cada uno de los módulos:

- **Cliente:** Como se ha descrito antes, el cliente mostrará una interfaz web en la cual se realizarán principalmente tres funciones: envío de trabajos al servidor, visualización de información del sistema y visualización del historial de trabajos enviados.

  En el apartado de envío de trabajos al servidor se especificarán todos los parámetros con los que se ejecutará el trabajo en la cola. En el apartado de visualización de información del sistema se mostraran distintos datos de los nodos disponibles, la salida del trabajo ejecutado e información de las colas disponibles. Finalmente en el apartado de visualización del historial, se mostrará un pequeño registro de los trabajo que se han ejecutado y sus parámetros.

- **Servidor:** Como también se ha descrito antes, el servidor se ejecutará en un sistema operativo Linux. Éste ofrecerá distintos servicios vía *WebServices* los cuales utilizará el cliente para enviar trabajos.

  El Servidor debe tener instalado el sistema de colas *Torque* que se utilizará para la ejecución de los diferentes trabajos que el servidor reciba.

3.4. ASPECTOS DE SEGURIDAD EN EL SISTEMA

El presente apartado tiene como finalidad realizar un breve análisis sobre las vulnerabilidades detectadas en el sistema. Cabe decir que, aunque se es consciente que toda aplicación informática debe ser segura por sí misma, no se han solventado dichas

vulnerabilidades debido a que la carga de trabajo que esto supondría, haría que el proyecto sobrepasara los valores de tiempo y presupuesto estimados. De esta manera, la solvencia de dichas fallas se ha propuesto en el apartado Trabajos futuros.

A continuación se describen estas vulnerabilidades y propuestas para solventar las mismas.

Vulnerabilidad	Robo de credenciales (usuario/contraseña)
Propuesta de solución	Cifrado de credenciales mediante un algoritmo seguro. Estableciendo un túnel SSL entre cliente y servidor, se logra una comunicación cifrada entre aplicación cliente y servidor. Como solución alternativa y más encilla se propone el cifrado de la contraseña con algún algoritmo de hashing estilo MD5 o SHA1 preferiblemente.

Tabla 20: Robo de credenciales

Vulnerabilidad	Visualización del trafico y ataque de hombre en medio
Propuesta de solución	La solución propuesta anteriormente, sería válida para solventar dicha vulnerabilidad ya que la información se transmitiría cifrada mediante el túnel SSL.

Tabla 21: Visualización del trabajo y ataque de hombre en medio

Vulnerabilidad	Ejecución de programas que revelen información no autorizada: /etc/shadow, etc...
Propuesta de solución	Limitar los permisos con los que se ejecutan las aplicaciones en el sistema.

Tabla 22: Ejecución de programas malintencionados

Vulnerabilidad	Inserción de comandos en formularios: estilo SQL Injection
Propuesta de solución	Filtrar los parámetros introducidos en los formularios. en concreto se deberían de filtrar caracteres como ‘,”,;.

Tabla 23: Inserción de comandos en formularios

Don't waste your life. No one chooses mediocracy but many settle for it.
NEVER SETTLE
(Unknown)

4. DISEÑO DEL PROYECTO

En este capítulo se especifica el modelo utilizado para la realización del sistema. Para esto, se han seguido las pautas que establece el estándar UML, de esta manera lograremos un diseño estricto y a la vez sencillo del sistema. Para la representación de la relación entre los distintos componentes se ha utilizado la herramienta *Rational Rose*.

Existen una serie de características de la aplicación que hay que establecer con exactitud, éstas son: el alcance de la aplicación, las tecnologías a utilizar para su realización y finalmente determinar el tipo de usuario al que va dirigido. Es muy importante establecer bien estos parámetros ya que la calidad del producto dependerá en gran medida de ello. Por esta razón, entre otras, es conveniente utilizar una metodología de diseño y desarrollo.

El objetivo principal del documento es determinar la arquitectura del sistema a desarrollar y su correspondiente división en componentes de bajo nivel. De este modo se dispone del sistema de información listo a ser desarrollado, pues debe existir una correspondencia uno a uno entre un componente de bajo nivel y un elemento software.

4.1. INTRODUCCIÓN A UML

UML es una especificación de notación orientada a objetos. Se basa en las anteriores especificaciones *BOOCH, RUMBAUGH* y *COAD-YOURDON*. Divide el proyecto en un

número de diagramas que representan las diferentes vistas del proyecto. La relación entre estos diagramas representa la arquitectura del proyecto.

Con UML nos debemos olvidar del protagonismo excesivo que posee el diagrama de clases, éste representa una parte importante del sistema, pero solo representa una vista estática, es decir no lo muestra en marcha. Sabemos su estructura pero no sabemos lo que le sucede a sus diferentes partes cuando comienza su funcionamiento.

UML introduce nuevos diagramas que representan una visión dinámica del sistema. Es decir, gracias al diseño de la parte dinámica del sistema podemos localizar problemas de la estructura o de las partes que necesitan ser sincronizadas, así como del estado de cada una de las instancias en cada momento. El diagrama de clases continua siendo muy importante, pero se debe tener en cuenta que su representación es limitada, y que ayuda a diseñar un sistema robusto con partes reutilizables, pero no a solucionar problemas de propagación de mensajes ni de sincronización o recuperación ante estados de error. En resumen, un sistema debe estar bien diseñado, pero también debe funcionar bien.

UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

UML es ahora un Standard, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otra rama.

UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica.

4.2. HERRAMIENTA DE DESARROLLO SOFTWARE

Se utilizará el entorno de trabajo específico de la plataforma .NET *Visual Studio 2008 Service Pack 1*, a la cual se le ha añadido las *tools de Silverlight,* para el desarrollo en esta tecnología. *Visual Studio* proporciona potentes editores para la codificación del sistema; además de proporcionar igualmente los compiladores, *debuggers* y *linkers* necesarios para llevar a cabo la construcción del sistema.

Para el diseño de la interfaz web se ha utilizado *Expression Blend 2* con las *tools de Silverlight*.

Para el desarrollo del servidor en Java, se ha utilizado el entorno de trabajo *Eclipse 3.5* con *Apache Axis & Tomcat* para el desarrollo de los Web Services. Éste entorno proporciona todas las facilidades para implementar y publicar Web Services en el lenguaje de programación Java.

Para realizar la documentación se utilizará la herramienta *Microsoft Office 2007*.

Los diagramas que aclaren el diseño del sistema y de su código fuente será *Altova Umodel 2009*.

Para otros diagramas como los de funcionamiento se ha utilizado *Microsoft Visio 2007*

4.3. CONVENCIONES DE NOMBRADO

A continuación se describen las convenciones de nombrado y codificación:

- Todas las clases del sistema empezarán siempre su nomenclatura por una letra mayúscula y el resto de su nomenclatura en minúscula, a excepción de las letras que se correspondan con el comienzo de una palabra, que irán de nuevo en mayúscula para identificar las palabras que forman el nombre de la clase; por este motivo sólo se aceptarán letras y números en su nomenclatura, nunca ningún otro carácter.
- Un método de una clase empezará siempre en su nomenclatura por una letra mayúscula y el resto de su nomenclatura en minúscula, a excepción de las letras

que se correspondan con el comienzo de una palabra, que irán de nuevo en mayúscula para identificar las palabras que forman el nombre del método; por este motivo sólo se aceptarán letras y números en su nomenclatura, nunca ningún otro carácter. Los atributos en cambio, empezarán todos por el carácter *'_'*, seguido de cualquier otro carácter.

- Para cada método y atributo se debe especificar siempre su visibilidad (pública privada…)
- Todos los atributos de una clase serán siempre privados.
- Todos los métodos de una clase serán siempre privados, a excepción de los métodos que necesiten ser llamados fuera de la propia clase.
- Todo el código fuente estará debidamente indentado para aumentar su comprensión.
- Se buscará siempre la sencillez y facilidad de compresión del código fuente.

4.4. MODELADO DE LA ARQUITECTURA ESTÁTICA

En el presente apartado se pretenden identificar los principales componentes del sistema en términos de Paquetes y Clases.

4.4.1. *Identificación de clases*

Analizando los requisitos identificados, a continuación se realizará una descripción sobre cada una de las clases que se han considerado. Para lograr una mejor organización de la aplicación, estas clases se han incluido en tres entidades que denominaremos paquetes:

- Interfaz web
  - **Page:** Esta clase contiene todos los elementos de la interfaz web, en ella se incluyen los componentes visuales como botones, cajas de texto, etc, así como los métodos que realizan las funciones principales de la interfaz como la captura de eventos o incluso el acceso a los Web Services.

- Clente srgc
    - **Jobs:** Esta clase guarda una lista de las tareas que se han lanzado al servidor. Cada elemento de dicha lista es un objeto tipo *Job*.

    - **Job:** En esta clase se guardan todos los datos relacionados con una tarea lanzada al servidor. Se incluye toda la configuración de la tarea, desde los parámetros básicos hasta el ejecutable que se ha lanzado.

    - **Queues:** Esta es una clase que almacena una lista de todas las colas disponibles en el sistema. Cada elemento de dicha cola es un objeto de la clase Queue.

    - **Queue:** Aquí se guardan todos los parámetros de configuración de cada cola existente en el Servidor.

    - **Jobs_queue:** Contiene una lista de todos los trabajos en ejecución en las colas de trabajo del Servidor.

    - **Job_queue:** Esta clase contiene información básica de los trabajos en ejecución en el Servidor. Aquí se implementan métodos para eliminar trabajos en ejecución de las colas de trabajo.

    - **Nodes:** Aquí se almacena una lista de los nodos disponibles en el cluster del Servidor. Cada elemento de dicha lista es un objeto de la clase Node.

    - **Node:** En esta clase se incluyen datos de información respecto a los nodos que forman el cluster del Servidor.

    - **NodeStatus:** En esta clase se almacena el estado en el que se encuentra cada nodo perteneciente al cluster. También se guardan datos sobre sus características mñas relevantes.

- Servidor srgc
  - **Server_srgc**: Clase principal de la aplicación. Aquí se incluyen todos los métodos que se ofrecen en forma de WebServices.
  - **Config_job:** Aquí se guarda la configuración del último trabajo que el servidor ha recibido para ejecución.
  - **Nodes_server: S**e almacenan datos de información respecto a los nodos que forman el cluster del Servidor.
  - **Queues_server:** Esta clase contiene una lista de objetos de la clase *Queue_server*.
  - **Queue_server:** En esta clase se almacenan los datos de las colas de ejecución creadas en el Servidor.
  - **Usuario:** Contiene las credenciales que se utilizarán para autenticar a un usuario legítimo del Sistema.

4.4.2. Identificación de atributos y métodos

A continuación se mostrarán las clases con sus respectivos atributos y métodos. Para los atributos se muestra información como el tipo de dato y la visibilidad (pública/privada). En cuanto a los métodos se ha indicado su visibilidad, los parámetros que acepta y el tipo de valor devuelto.

La Ilustración 11: Paquetes de la aplicación, muestra un diagrama general de los paquetes de la aplicación.

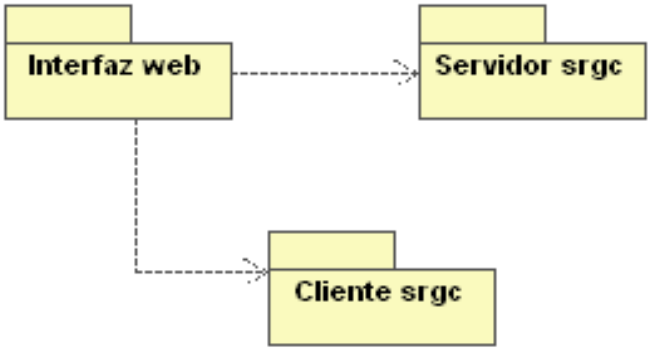


Ilustración 11: Paquetes de la aplicación

Clases en el cliente

- **Clase Page**

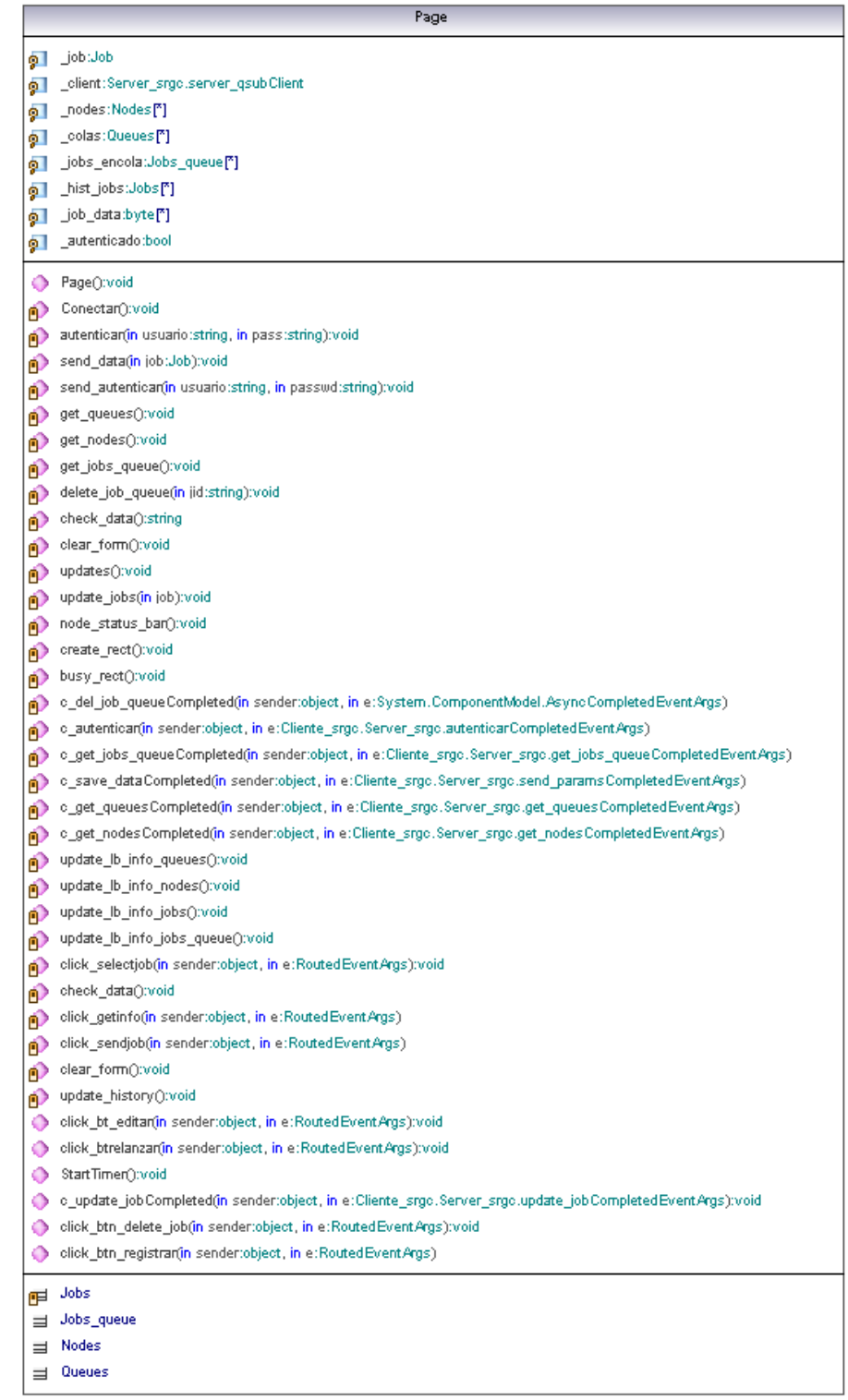


Ilustración 12: Clase Page

- **Clase Jobs**

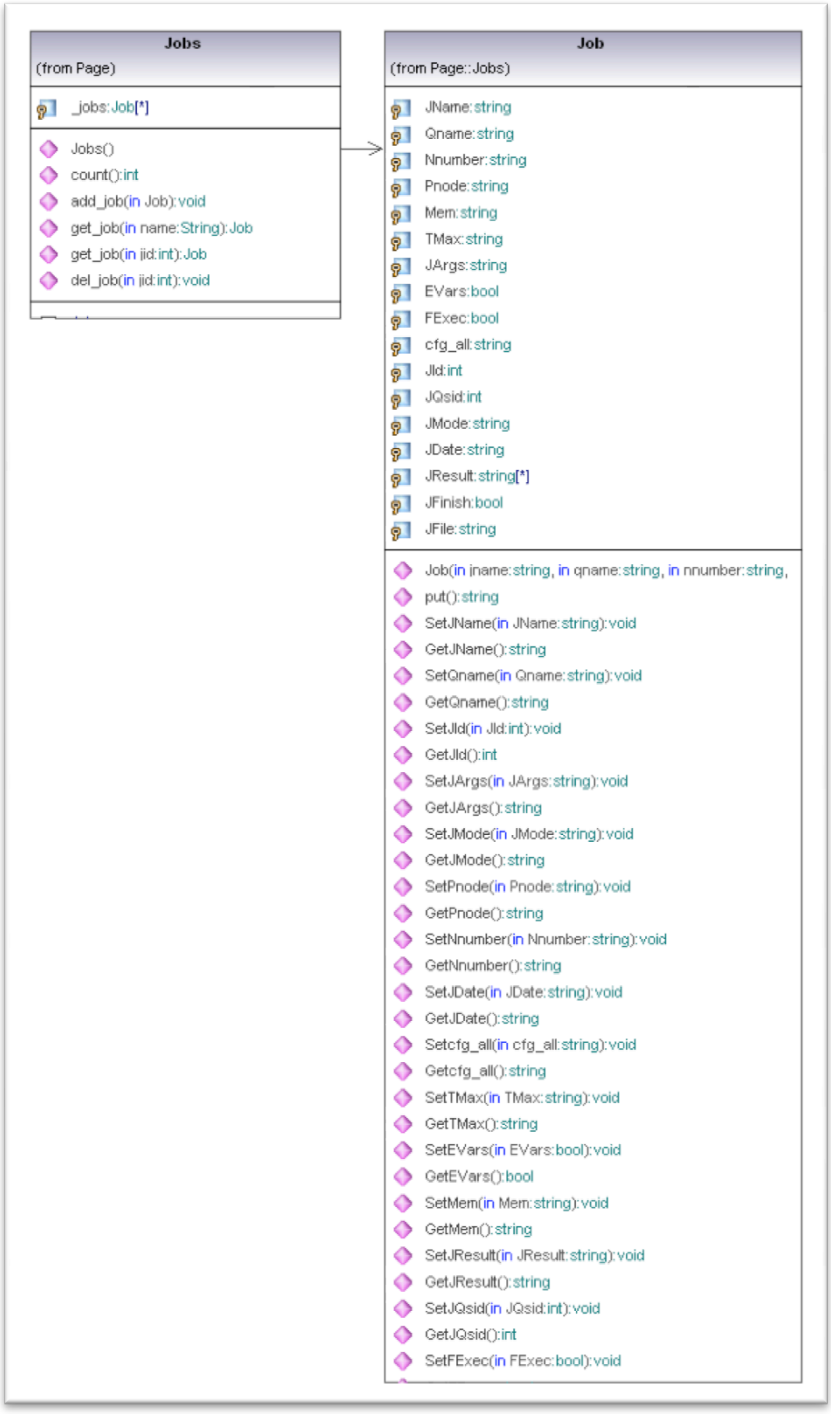


Ilustración 13: Clase Jobs

- **Clase Nodes**

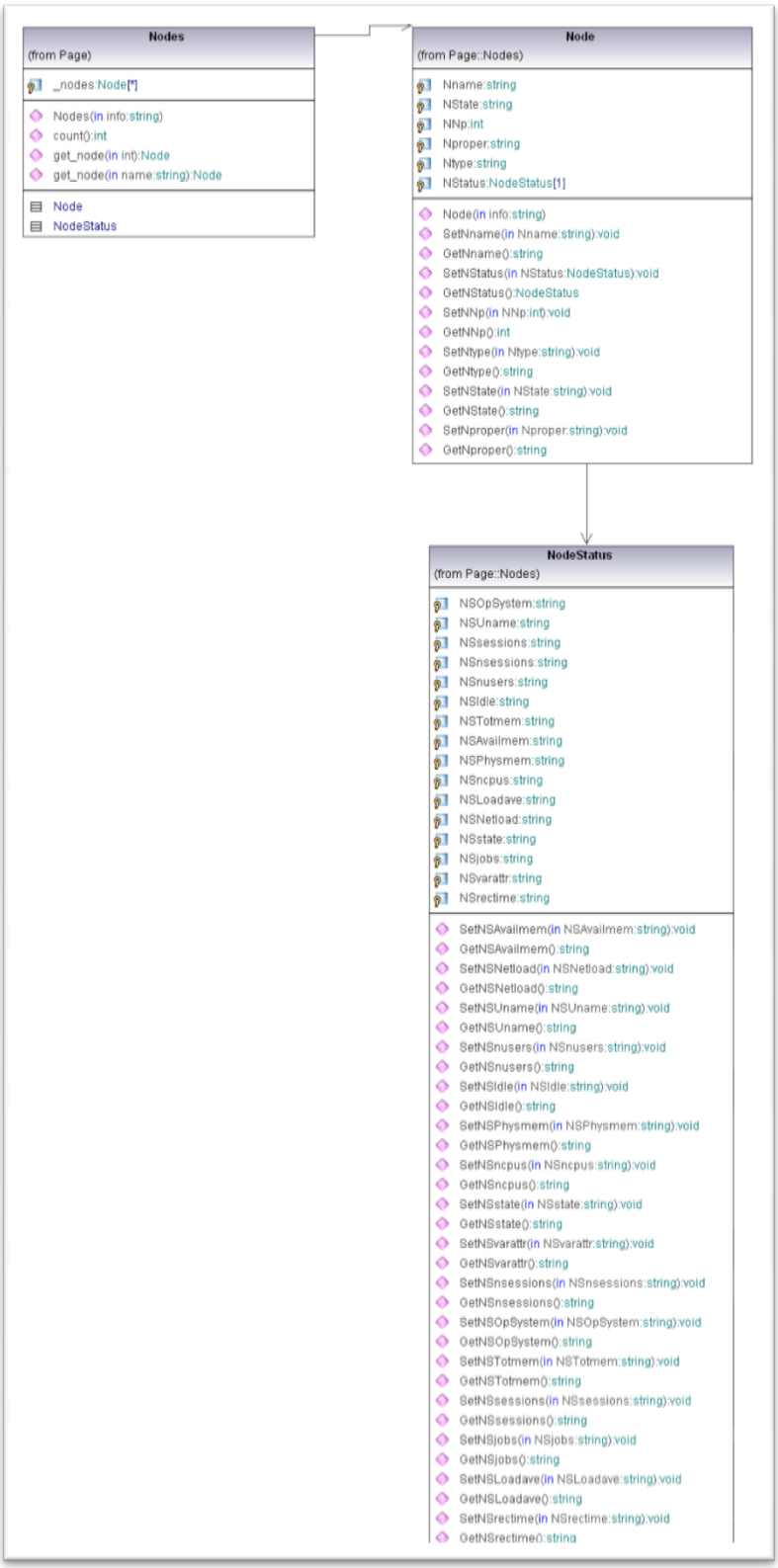


Ilustración 14: Clase Nodes

- **Clase Queues**

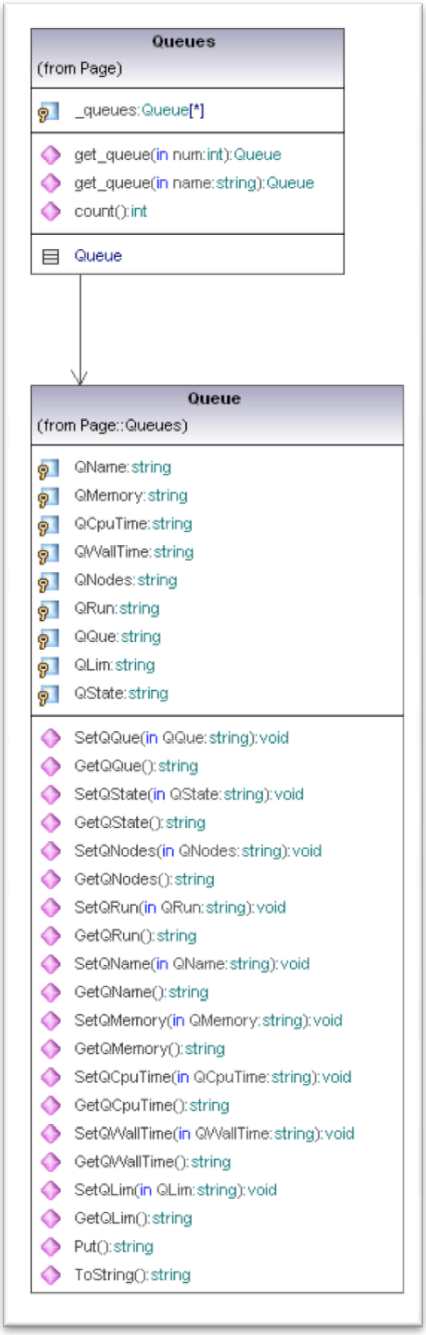


Ilustración 15: Clase Queues

- **Clase Jobs_queue**

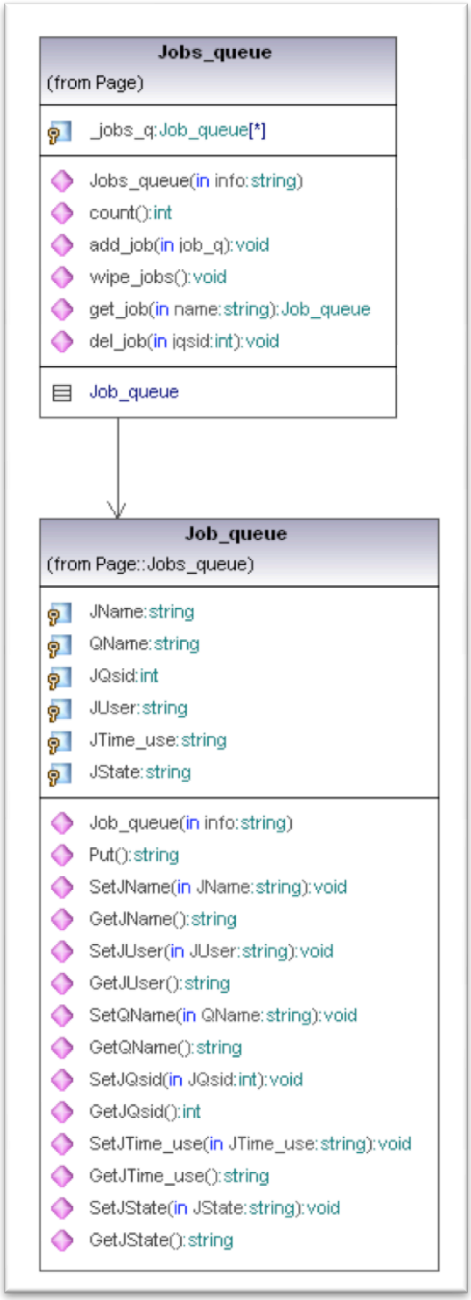


Ilustración 16: Clase Jobs_queue

Clases en el Servidor

- **Clase server_srgc**

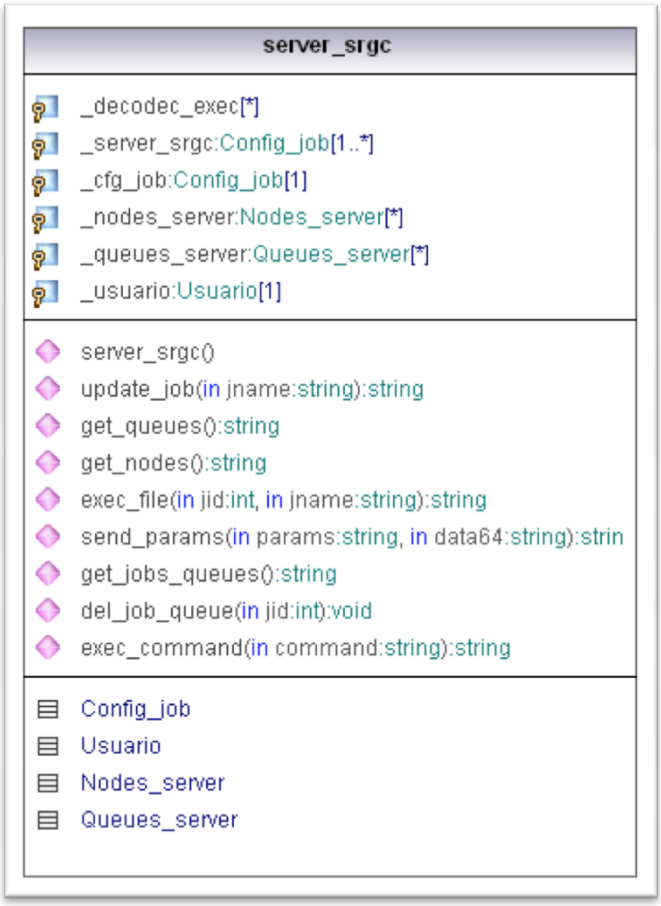


Ilustración 17: Clase server_srgc

- **Clase Config_job**

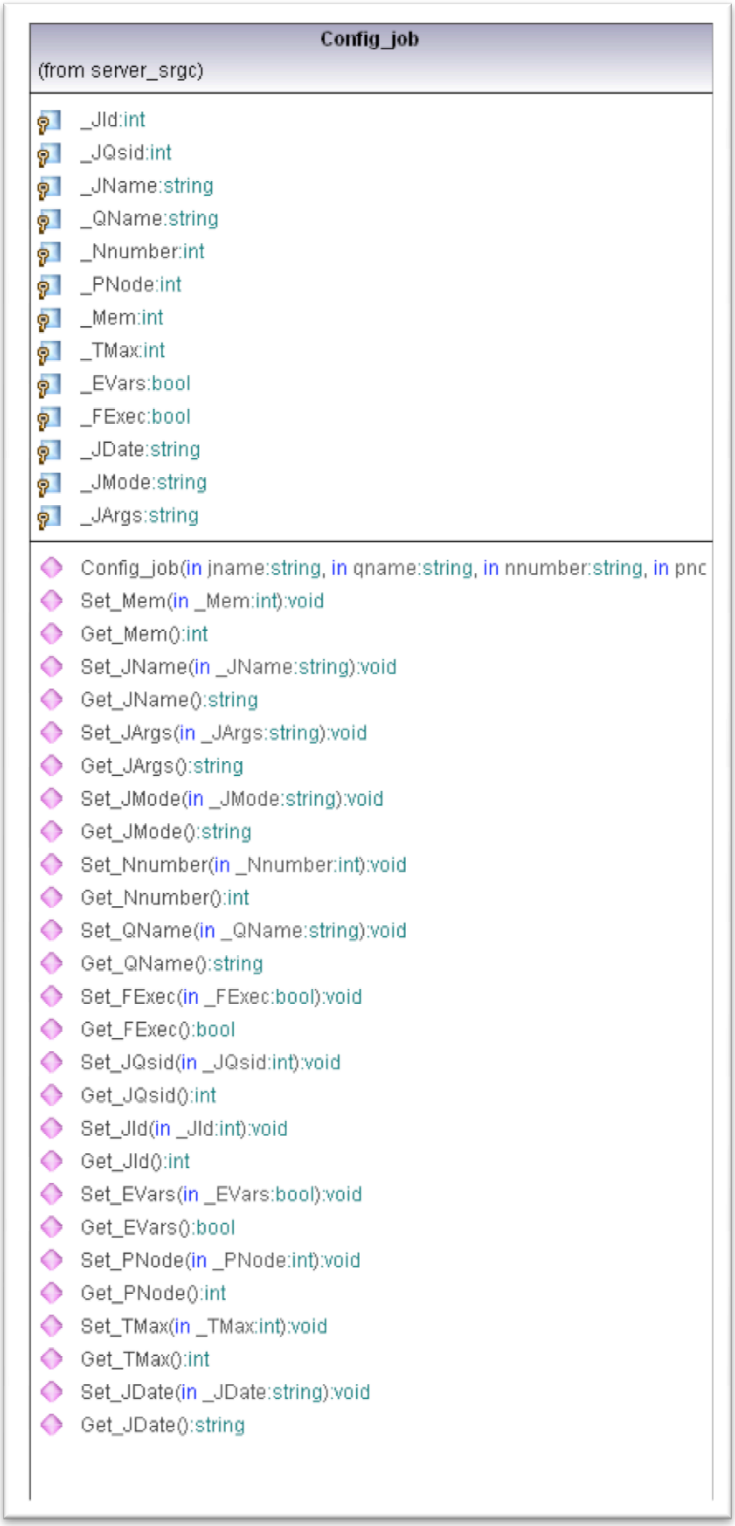


Ilustración 18: Clase config_job

- **Clase Nodes_server**

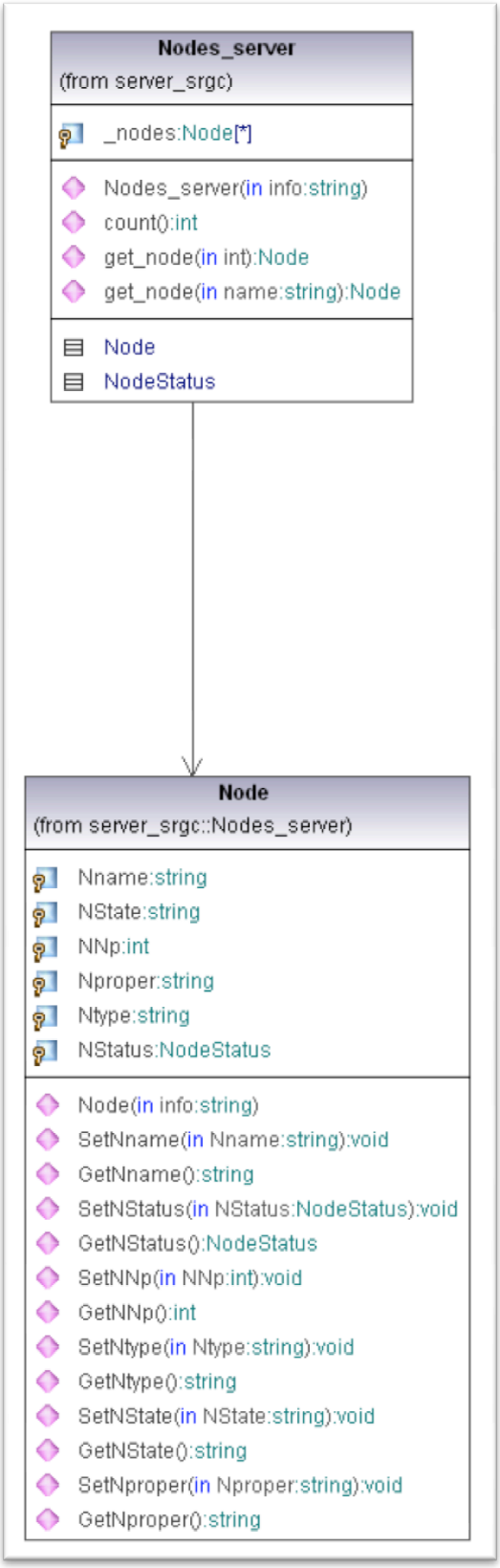


Ilustración 19: Clase Nodes_server

- **Clase Queues_server**

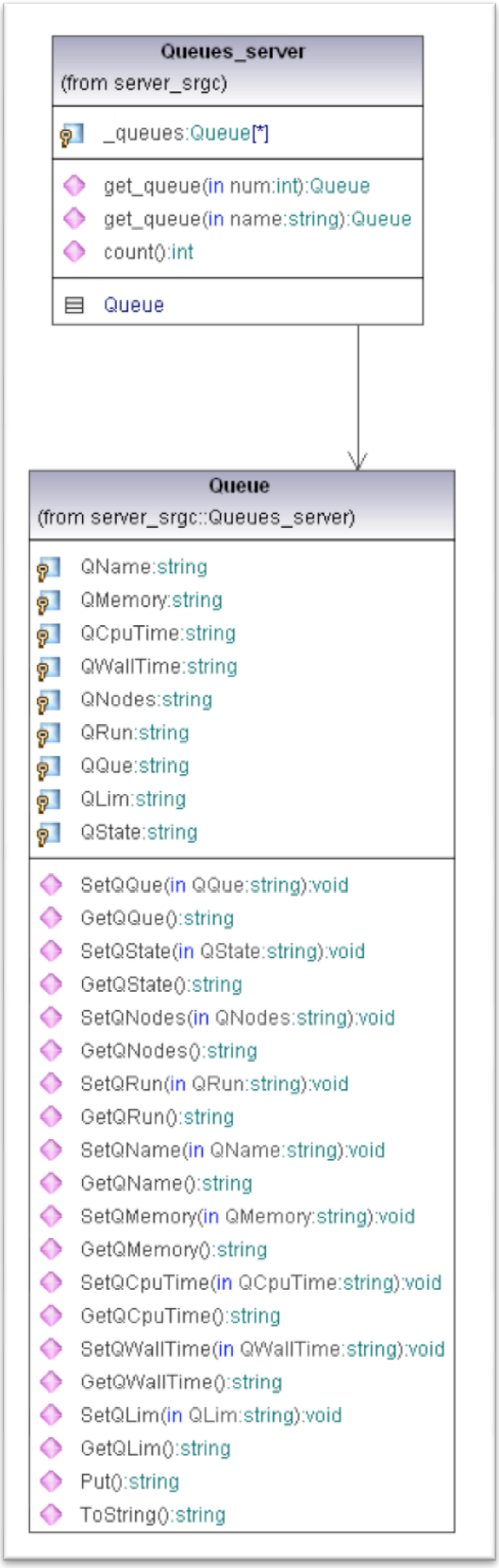


Ilustración 20: Clase Queues_server

- **Clase Usuarios**

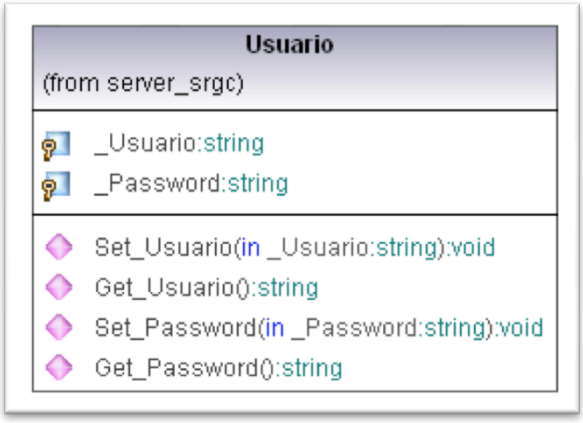


Ilustración 21: Clase Usuarios

4.4.3. Diagrama de clases

A continuación se muestran los diagramas de Clases de la Interfaz de usuario Web y del Servidor respectivamente.

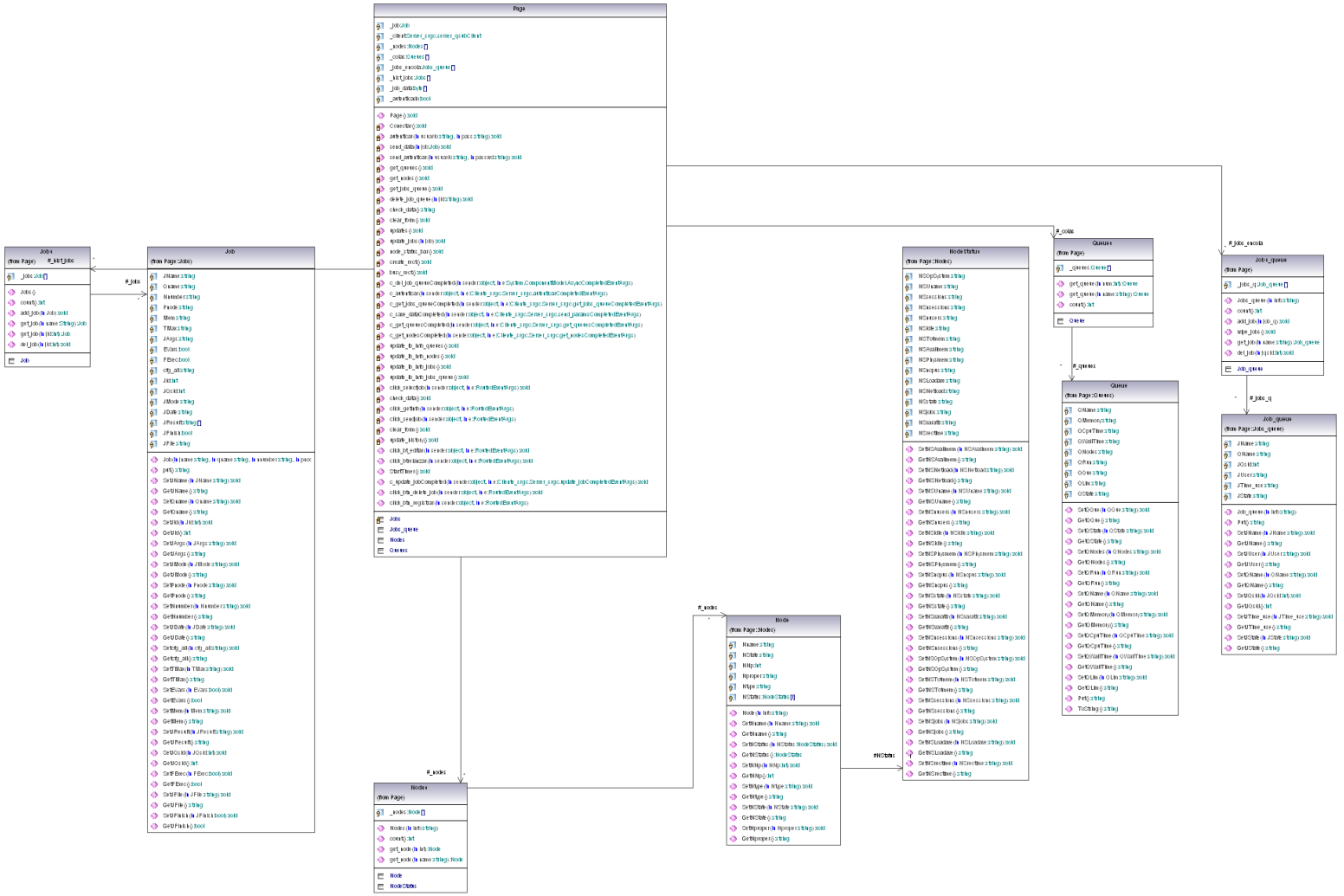


Ilustración 22: Diagrama de clases de la interfaz web

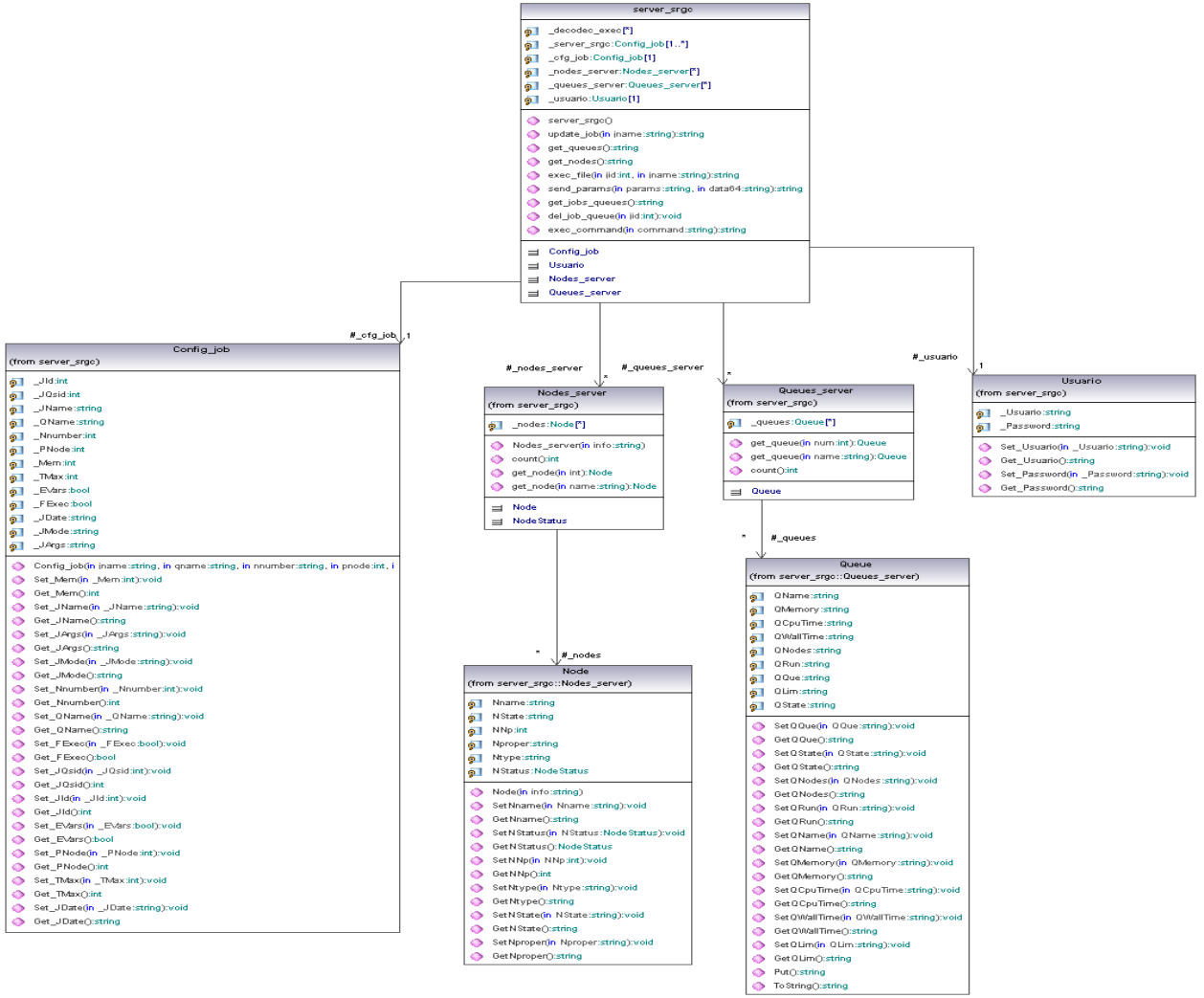


Ilustración 23: Diagrama de clases del Servidor

4.4.4. Archivo de disposición de la Interfaz de Usuario en XAML (Page.xaml)

Dado que la interfaz de usuario de la aplicación cliente está programada con la tecnología Silverlight, ésta utiliza un lenguaje declarativo basado en XML llamado XAML.

La declaración de los objetos que serán visibles en la interfaz de usuario, se realiza en XAML y en el caso del presente proyecto, en el fichero Page.xaml. Una vez declarados todos los elementos, en las diferentes clases implementadas se interactua con los mismos. Debido al tamaño de este fichero, se procede a adjuntar su contenido junto al resto de código presentado con este documento.

4.5. MODELADO DE LA ARQUITECTURA DINÁMICA

En este apartado se incluirán detalles de las partes dinámicas del sistema. En concreto se mostrarán diagramas de caso de uso, diagramas de estado y diagramas de secuencia.

4.5.1. Diagrama de casos de uso

Los diagramas de casos de uso se emplean para comprender como se comporta el sistema ante las diferentes situaciones que puedan surgir en la ejecución de la aplicación. En ellos no se especifica cómo se hace, sino qué hace el sistema ante las diferentes situaciones.

Los diagramas de casos de uso también resultan útiles para facilitar la comunicación entre los expertos de dominio y los informáticos que desarrollarán finalmente la aplicación sin llegar a un nivel de detalle exhaustivo.

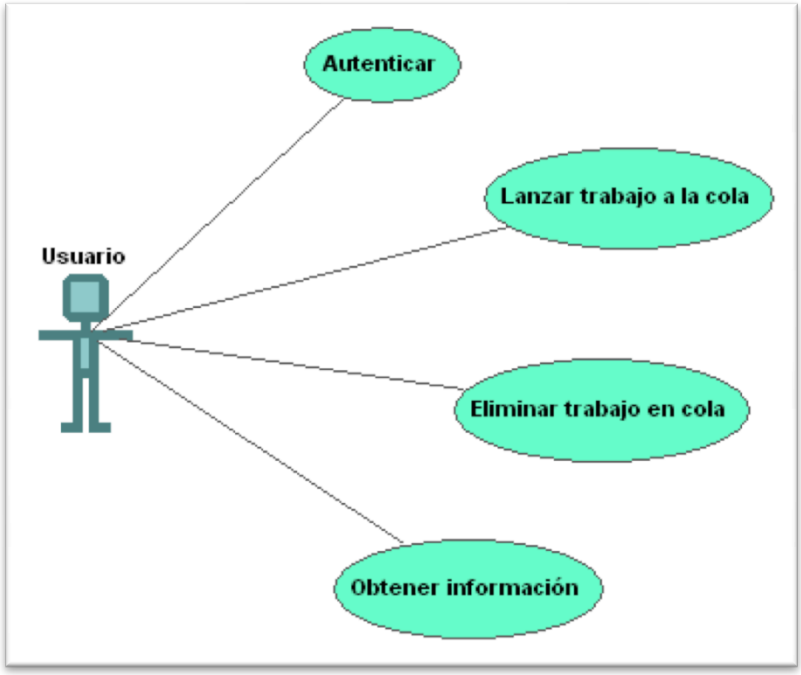


Ilustración 24: Diagrama de casos de uso general

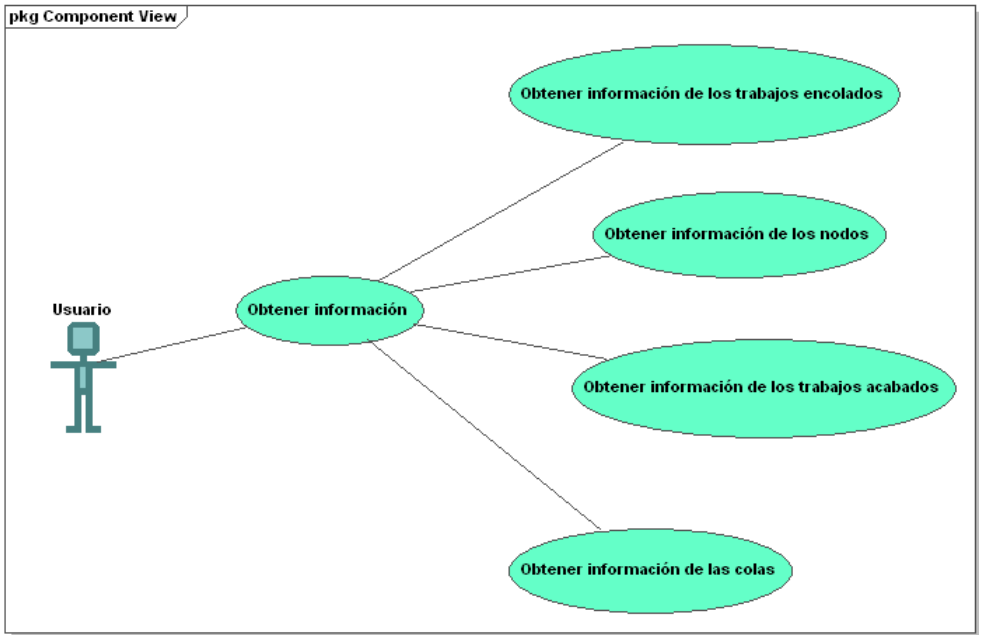


Ilustración 25: Diagrama del caso de uso Obtener información

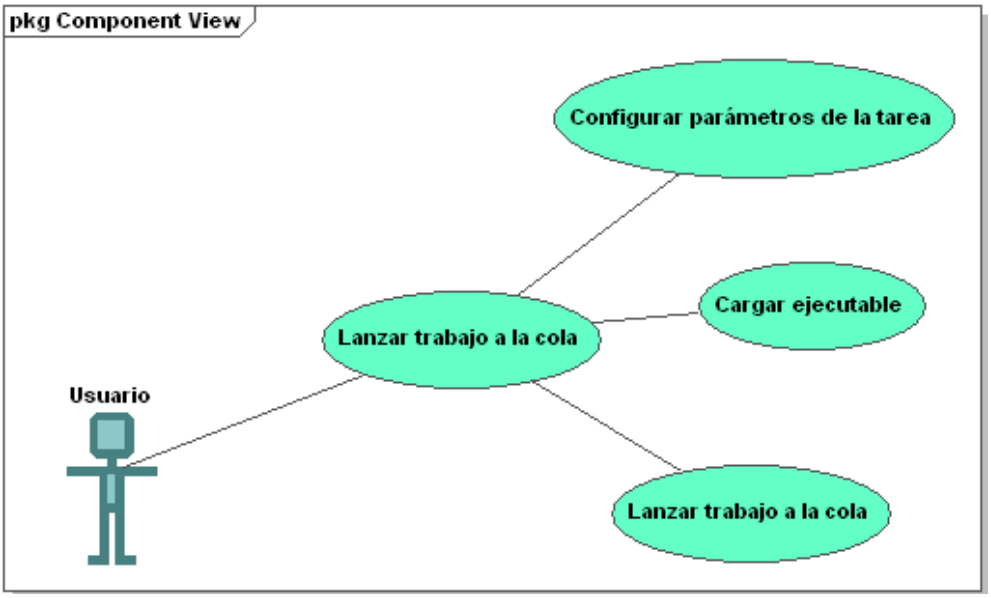


Ilustración 26: Diagrama del caso de uso Lanzar trabajo a la cola

4.5.2. Diagrama de estados

A continuación se incluirá un diagrama de estados en el que se incluyen las partes dinámicas del sistema. Este diagrama modela los estados en los que el sistema se encuentra en todo momento dependiendo de las acciones que el usuario realice con el mismo. Se incluyen los estados, las acciones y las respuestas del sistema.

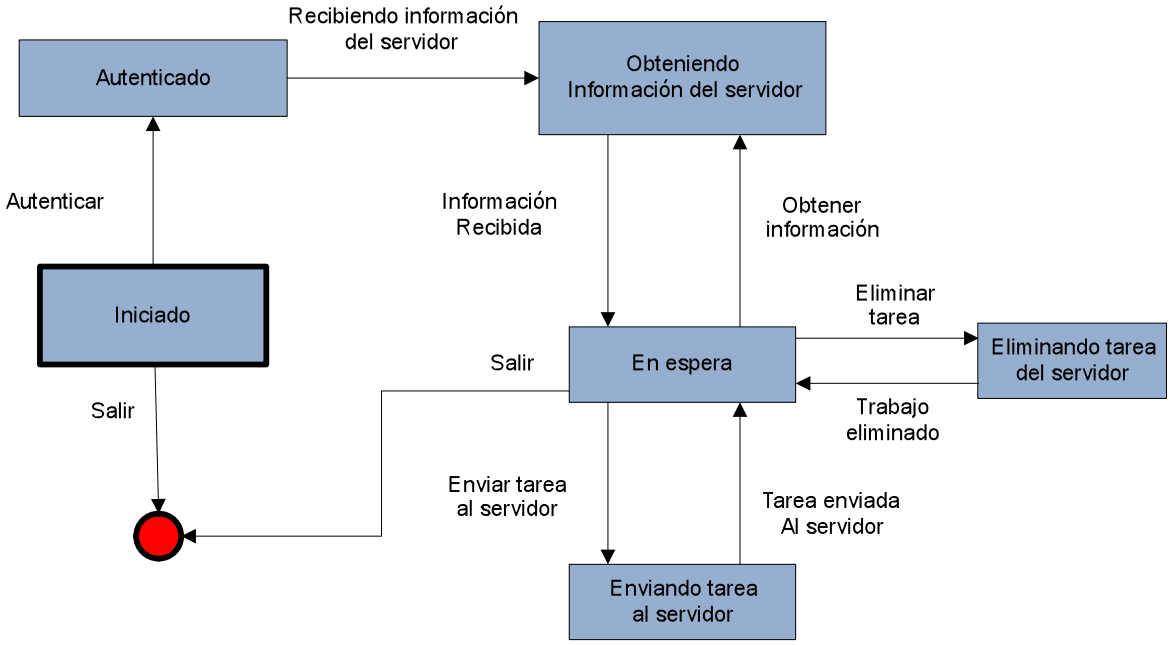


Ilustración 27: Diagrama de estados

4.5.3. Diagrama de secuencia

A continuación, por cada caso de uso obtenido en el apartado anterior, se muestra un diagrama de secuencia en el cual se puede apreciar el comportamiento del mismo y así obtener una visión mas clara de la aplicación.

- **Autenticarse**

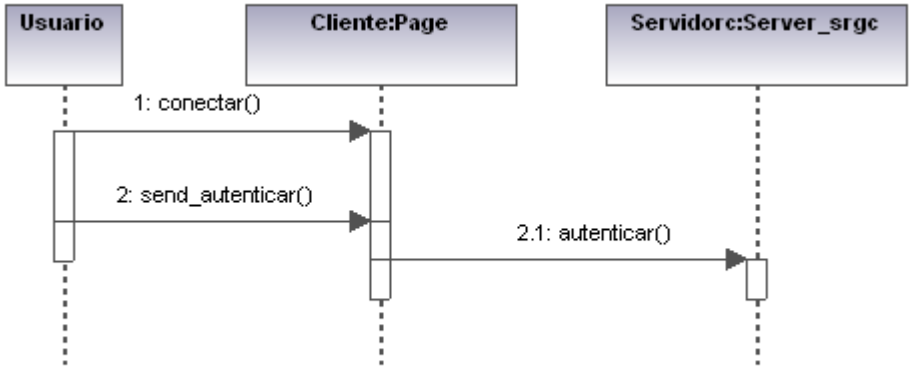


Tabla 24: Diagrama de secuencia: Autenticación

- **Configurar parámetros**

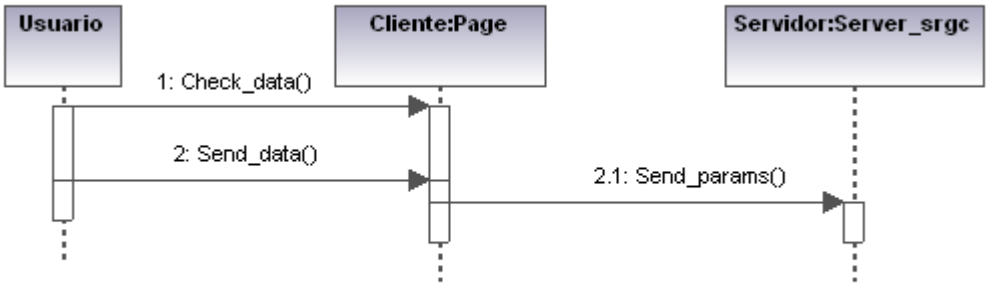


Tabla 25: Diagrama de secuencia: Configuración de parámetros

- **Cargar ejecutable**

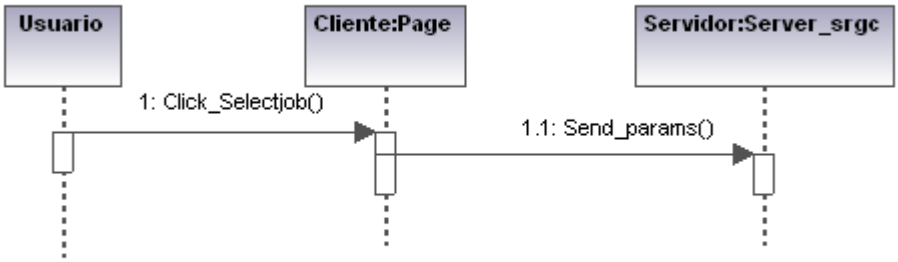


Tabla 26: Diagrama de secuencia: Cargar ejecutable

- **Borrar trabajo de la cola**

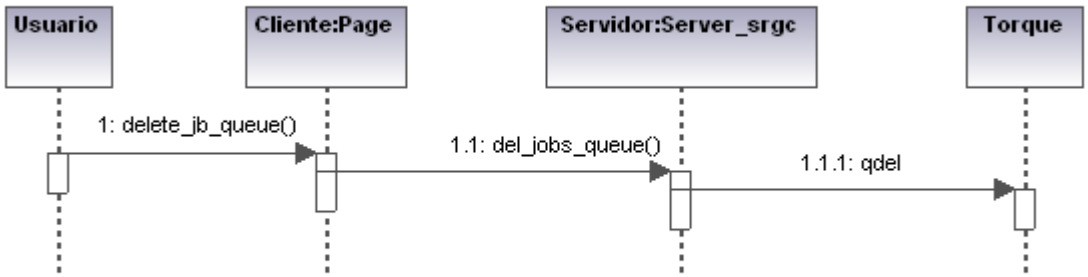


Tabla 27: Diagrama de secuencia: Borrar trabajo de la cola

- **Lanzar trabajo**

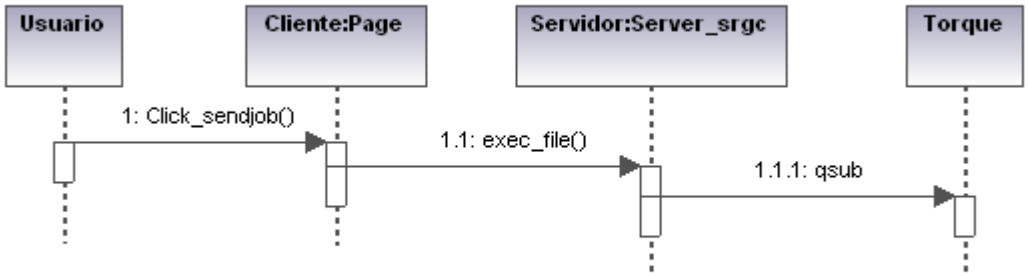


Tabla 28: Diagrama de secuencia: Lanzar trabajo

- **Obtener información de trabajos encolados**

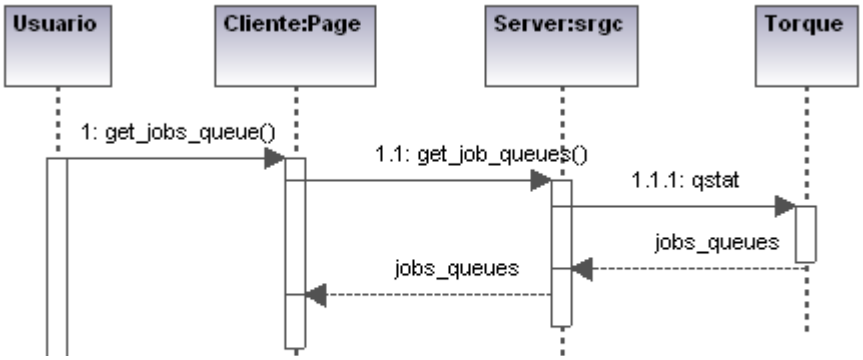


Tabla 29: Diagrama de secuencia: Obtener información de los trabajos encolados

- **Obtener información de trabajos finalizados**

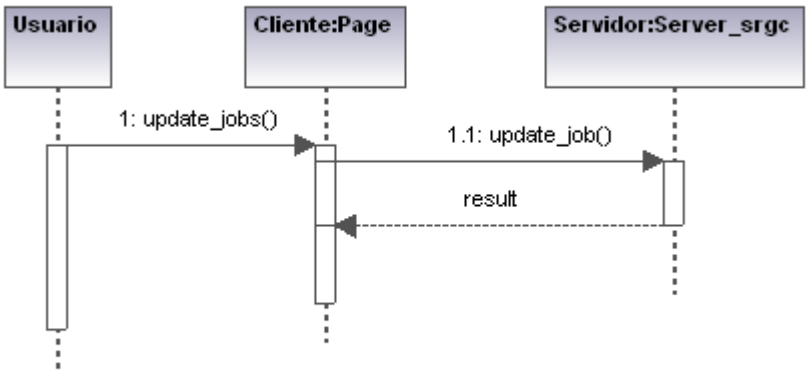


Tabla 30: Obtener información de trabajos finalizados

- **Obtener información de los nodos**

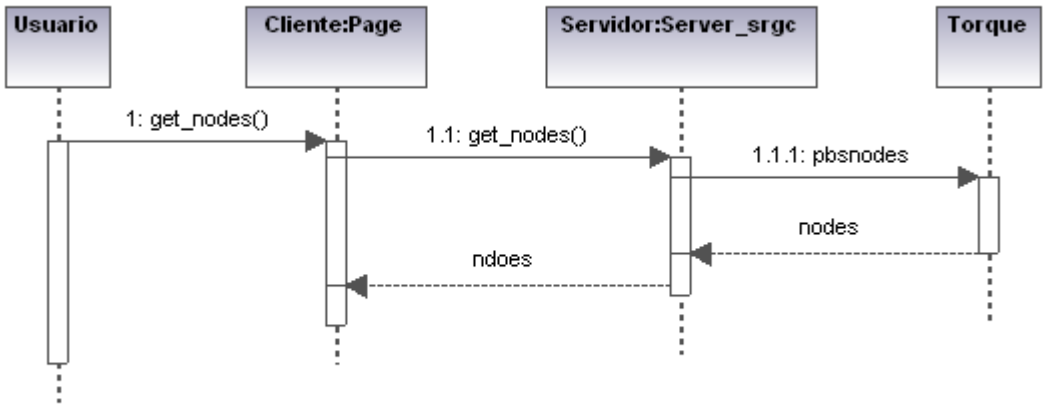


Tabla 31: Obtener información de los nodos

- **Obtener información de las colas de trabajo**

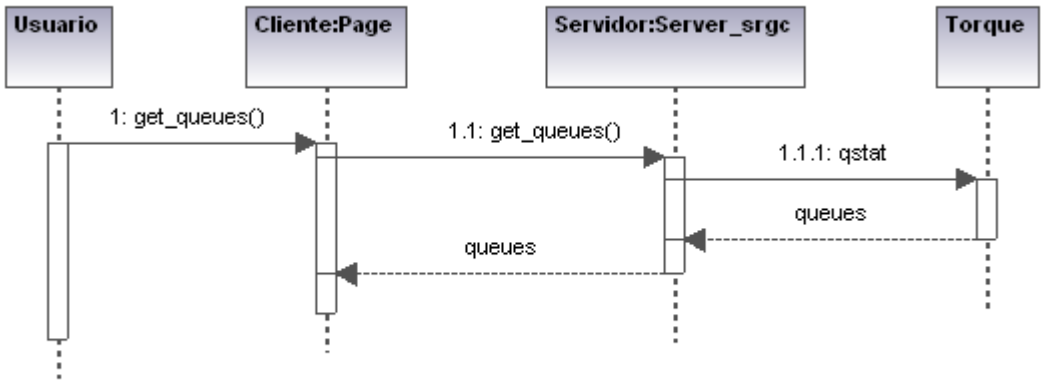


Tabla 32: Obtener información de las colas de trabajo

Se hace camino al andar y al volver al vista atrás se ve la senda que nunca se ha de volver a pisar.
(Antonio Machado)

5. PRUEBAS DEL SISTEMA

A continuación se proponen un conjunto de operaciones a realizar en la aplicación las cuales nos permitirán verificar que las funcionalidades requeridas en la aplicación se realizan satisfactoriamente.

Finalmente, se adjunta una matriz de trazabilidad que relaciona cada una de las pruebas descritas con los requisitos detectados en el apartado 3.3.

5.1. ESPECIFICACIÓN DE LAS PRUEBAS

Identificador	PR-01
Descripción	Comprobar que la autenicación se realiza correctamente con unas credenciales legítimas. De la misma forma comprobar que la autenticación no se realiza correctamete con unas credenciales no legítimas.
Resultado	Satisfactorio

Tabla 33: PR-01

Identificador	PR-02
Descripción	Verificar que una vez autenticado, la conexión con el servidor se realiza correctamente visualizando al barra de estado de los nodos en la parte inferior de la aplicación
Resultado	Satisfactorio

Tabla 34: PR-02

Identificador	PR-03
Descripción	Verificar que la información sobre los nodos que forman el cluster se obtiene del servidor.
Resultado	Satisfactorio

Tabla 35: PR-03

Identificador	PR-04
Descripción	Verificar que la información sobre las colas de ejecución se obtiene del servidor.
Resultado	Satisfactorio

Tabla 36: PR-04

Identificador	PR-05
Descripción	Verificar que la información sobre los trabajos encolados si los hubiera, se obtiene del servidor.

Resultado	Satisfactorio

Tabla 37: PR-05

Identificador	PR-06
Descripción	Verificar que no es posible insertar datos que no sean de tipo entero en los campos de Memoria, Tiempo Max., Nodos, Proc./Nodos en la configuración de parámetros de jecución de la tarea.
Resultado	Satisfactorio

Tabla 38: PR-06

Identificador	PR-07
Descripción	Comprobar que no es posible lanzar un trabajo con el mismo nombre que otro trabajo lanzado anteriormente.
Resultado	Satisfactorio

Tabla 39: PR-07

Identificador	PR-08
Descripción	Comprobar que es posible rellenar todos los parámetros del trabajon y el envio de éstos al servidor se realiza correctamente.
Resultado	Satisfactorio

Tabla 40: PR-08

Identificador	PR-09
Descripción	Verificar que es posible realizar correctamente la selección del fichero ejecutable a lanzar en el servidor.
Resultado	Satisfactorio

Tabla 41: PR-09

Identificador	PR-10
Descripción	Comprobar que una vez hemos lanzado la ejecución de un trabajo al servidor, éste aparece en el apartado *"Historial"*.
Resultado	Satisfactorio

Tabla 42: PR-10

Identificador	PR-11
Descripción	Comprobar que mientras un trabajo se encuentra en ejecución, éste se muestra en *"Trabajos encolados"*.
Resultado	Satisfactorio

Tabla 43: PR-11

Identificador	PR-12
Descripción	Comprobar que una vez terminada la ejecución del trabajo enviado, éste muestra tanto su información de salida como los errores en la pestaña *"Trabajos Acabados"*.

Resultado	Satisfactorio

Tabla 44: PR-12

Identificador	PR-13
Descripción	Comprobar que es posible eliminar un trabajo de la cola de ejecución pulsando en el botón *"Eliminar"* en la pestaña de *"Trabajos encolados"*.
Resultado	Satisfactorio

Tabla 45: PR-13

Identificador	PR-14
Descripción	Comprobar que al pulsar el botón *"Editar"* de la pestaña *"Historial"*, los parámetros del tabajo seleccionado se muestran en la pestaña *"Lanzar"* para poder ser modificados.
Resultado	Satisfactorio

Tabla 46: PR-14

Identificador	PR-15
Descripción	Comprobar que al pulsar el botón *"Relanzar"* de la pestaña *"Historial"*, el trabjo seleccionado se vuelve a enviar al servidor para su ejecución.
Resultado	Satisfactorio

Tabla 47: PR-15

Identificador	PR-16
Descripción	Comprobar que la aplicación actualiza periódicamente el estado de los nodos, las colas disponibles y los trabajos tanto encolados como acabados.
Resultado	Satisfactorio

Tabla 48: PR-16

Identificador	PR-17
Descripción	Comprobar que la aplicación permite el acceso simultáneo a más de un usuario a la vez.
Resultado	Satisfactorio

Tabla 49: PR-17

5.2. MATRIZ DE TRAZABILIDAD: REQUISITOS SOFTWARE VS PRUEBAS

	PR-01	PR-02	PR-03	PR-04	PR-05	PR-06	PR-07	PR-08	PR-09	PR-10	PR-11	PR-12	PR-13	PR-14	PR-15	PR-16	PR-17
URD-C-001			X	X	X			X	X	X	X	X	X				
URD-C-002																X	
URD-C-003		X															
URD-C-004						X		X									
URD-C-005									X								
URD-C-006			X														
URD-C-007					X												
URD-C-008				X													
URD-C-009										X							
URD-C-010															X		
URD-C-011														X			
URD-C-012																	X

Tabla 50: Matriz de trazabilidad

Tempus omnia fert, sed et aufert omnia tempus
(Anónimo)

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se exponen los objetivos descritos al principio del proyecto y el grado de cumplimiento de los mismos. Finalmente se proponen también, futuras mejoras de la aplicación orientada a la gestión del sistema de colas.

6.1. CONCLUSIONES

En este apartado se procede a evaluar el cumplimiento de los objetivos marcados en la introducción. Después de analizar dichos la aplicación realizada, teniendo en cuanta dichos objetivos, llegamos a la conclusión de que la aplicación cumple perfectamente con lo descrito, algunas de las funcionalidades llevadas a cabo que se consideran más importante son:

- El usuario es capaz de especificar cada uno de los parámetros considerados, con los que se ejecutará el trabajo en la cola.
- La ejecución remota de programas se realiza correctamente, ejecutándose en la cola seleccionada el programa compilado indicado por el usuario.
- El usuario, puede visualizar en tiempo real el estado de los nodos que forman el cluster.

- Tras realizarse algunas pruebas del sistema bajo condiciones de alta carga, el comportamiento ha sido es esperado en todo momento.
- Para la comunicación entre cliente y servidor, se han utilizado llamadas Web Service. Éstos proporcionan un servicio multiplataforma lo cual nos ha permitido realizar la implementación de cliente y servidor con tecnologías totalmente diferentes.
- El servidor permite atender concurrentemente las peticiones de múltiples clientes.

A parte de los objetivos marcados al inicio del proyecto, se han añadido otras características a la aplicación que inicialmente no se habían tenido en cuenta:

- Se visualizan los trabajos ya existentes en la cola del servidor previo a la ejecución de la aplicación cliente.
- Es posible la eliminación de trabajos encolados.
- Previamente a interactuar con el servidor, es necesario una autenticación por parte del cliente. Las credenciales deben existir en la aplicación servidor.

La realización del mismo también ha sido útil para el aprendizaje de algunas tecnologías que desconocía y que hoy en día se utilizan en multitud de proyectos informáticos. A continuación se enumeran algunas de ellas.

- Estudio del funcionamiento de las arquitecturas distribuidas basadas en Servicios Web.
- Creación de *Servicios Web* en Linux utilizando Java .
- Publicación de Servicios Web mediante Apache Axis.
- Acceso *Servicios Web* desde el lenguaje de programación *C#.*
- Implementación de una aplicación con la tecnología *Silverlight*.
- Conocimiento del lenguaje *C#* y del entorno de programación *Visual Studio .Net* y *Expression Blend*.
- Creación y configuración del sistema de colas de ejecución *TORQUE*.

6.2. TRABAJOS FUTUROS

A continuación se proponen una serie de aspectos que en posibles mejoras de la aplicación se podrían tener en cuenta:

- Realizar una interfaz web para dispositivos móviles. De esta manera seria posible visualizar el estado del cluster en cualquier sitio donde se disponga de una conexión a internet. Como alternativa, también se podría implementar una pequeña aplicación para teléfonos con Sistema Operativo Android.
- Mejorar el aspecto de seguridad tal y como se ha descrito en el análisis.
- Autenticar los usuarios legítimos contra un servidor de tipo LDAP. Esta ampliación facilitaría la gestíon de los usuarios que lanzan los trabajos.
- Soporte de diferentes tipos de colas. Torque no es el único sistema de colas en el mercado. Se propone estandarizar la interfa para que se pueda dar sopote a distintos sistemas, entre ellos Sun Grin Engine y Condor.

6.3. MÉTODO DE TRABAJO Y PRESUPUESTO

En esta sección se describe la metodología utilizada en el proyecto así como la organización y planificación de tareas que se va a llevar a cabo en el proyecto; además, se especifican tanto los recursos humanos y materiales como la tecnología necesaria para el éxito del proyecto, y para finalizar, se da un presupuesto que refleja el coste total que supondrá el desarrollo del proyecto.

6.3.1. Método de trabajo

Para el desarrollo del proyecto se ha utilizado el ciclo de vida incremental o de versiones sucesivas. La elección de este ciclo de vida no se debe a que el proyecto precisase de múltiples entregas de software o fuese de gran tamaño, sino porque algunas de las funcionalidades del proyecto deben estar operativas antes que otras y además, este método permite validar el sistema a medida que se construye.

Con este método, cada versión o fase es un sistema funcional capaz de realizar progresivamente la función deseada.

Como se puede observar (Ilustración 28: Ciclo de vida), el ciclo de vida incremental posee una fase de análisis inicial y varias fases de diseño, implementación y evaluación incrementales.

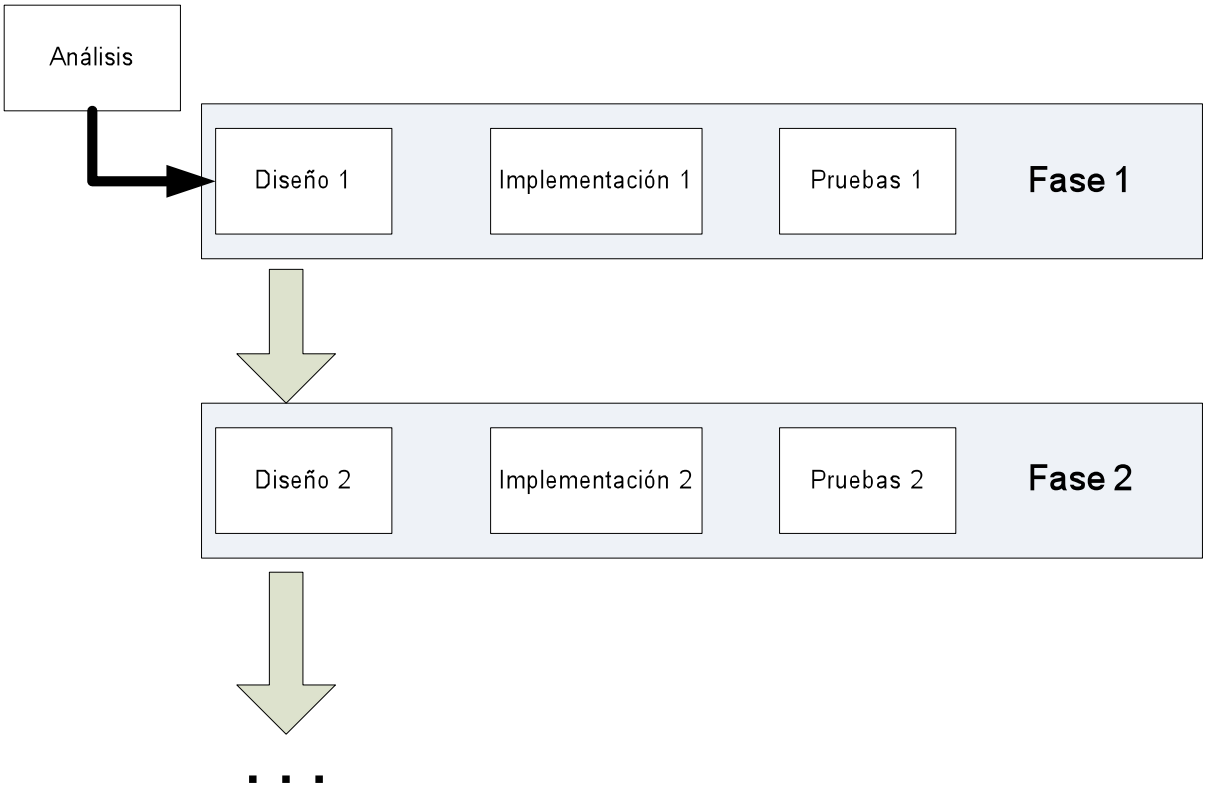


Ilustración 28: Ciclo de vida

A continuación se resumen los objetivos de las fases del diseño incremental.

- **Fase 1**: Diseño e implementación de la aplicación. En un principio solo aborda un prototipo cliente-servidor. Se implantan las clases fundamentales.

- **Fase 2**: Se añaden varias funcionalidades al simulador que no se planificaron en primera instancia.

- **Fase 3**: Se añade la concurrencia en el sistema. A partir de este momento, el servidor es capaz de atender simultáneamente varias peticiones de codificación y publicación.

- **Fase 4:** Se perfila la configuración de todos los aspectos del sistema mediante ficheros XML que permiten una sencilla manipulación.

- **Fase 5**: Se integra la generación de estadísticas mediante gráficos en tiempo real y la posibilidad de exportar dichos datos a un fichero de texto.

- **Fase 6**: Se termina de documentar el proyecto, finalizando aquellas partes que estuvieran inacabadas. En esta fase se realizan las pruebas de implantación del sistema, verificando que las librerías necesarias pueden ser accedidas.

6.3.2. Presupuesto

Gastos de personal imputables al proyecto

Para realizar el cálculo del presupuesto se han tenido en cuenta las siguientes consideraciones:

- Jornada laboral de 5 horas.
- 5 días laborables a la semana (excluidos fines de semana y festivos).
- Las vacaciones y días festivos considerados en la planificación son:
  - Festivos:
    - 5 días de Agosto.
    - 12 de Octubre: día de la hispanidad.
- Periodo de realización:
    - Inicio del proyecto: 1 de Julio.
    - Finalización del proyecto: 23 Noviembre.
- El proyecto requiere un analista y un programador.
- Coste de personal por hora de trabajo:
  - Analista: 40 €/hora.
  - Analista-programador: 25 €/hora.

Por lo tanto, el cómputo total de horas dedicadas al proyecto es 410 horas.

ACTIVIDADES	DURACIÓN (HORAS)	RECURSOS	TOTAL (EUROS)
ESTUDIO PREVIO	50	Analista	2.000

ANÁLISIS	100	Analista	4.000
FASE 1	10	Analista-Programador	250
FASE 2	40	Analista-Programador	1.000
FASE 3	50	Analista-Programador	1.250
FASE 4	30	Analista-Programador	750
FASE 5	20	Analista-Programador	500
FASE 6	80	Analista-Programador	2000
DOCUMENTACIÓN	30	Analista	1.200
TOTAL	**410 HORAS**		**12.950 €**

Tabla 51: Horas dedicadas

Costes de licencias para el desarrollo del sistema

- Visual Studio .Net 2008 600€.

Costes del hardware para el desarrollo del sistema

El sistema ha sido desarrollado utilizando este hardware:

- Intel Core 2 Duo 2,0GHz.
- 2 GBytes de RAM.
- 200 GBytes de almacenamiento.

Coste total del proyecto

La siguiente tabla detalla el coste total del proyecto:

CONCEPTO	COSTE (€)
RECURSOS HUMANOS	12.950
LICENCIAS SOFTWARE	600
TOTAL	**13.550 €**

Tabla 52: Coste total

El presupuesto total del proyecto es 34925 €.

Qui tegit veritatem, eam timet.
(Anónimo)

7. BIBLIOGRAFÍA

- **Libros**

[Moroney,2009] Silverlight 2.0

- **Referencias web**

[1] Introduccion .NET HTTP://WWW.DEVJOKER.COM/CONTENIDOS/CONCEPTOS-GENERALES-NET/88/INTRODUCCI%C3%B3N-A-NET.ASPX

[2] C Sharp Net HTTP://ES.WIKIBOOKS.ORG/WIKI/C_SHARP_NET

[3] Silverlight 2.0 (MSDN)

HTTP://MSDN.MICROSOFT.COM/ES-ES/LIBRARY/CC838158%28VS.95%29.ASPX

[4] Calling web services with Silverlight 2

HTTP://TIMHEUER.COM/BLOG/ARCHIVE/2008/03/14/CALLING-WEB-SERVICES-WITH-SILVERLIGHT-2.ASPX

[5] Consuming ASMX Web Services with Silverlight 2

HTTP://WWW.SILVERLIGHTSHOW.NET/ITEMS/CONSUMING-ASMX-WEB-SERVICES-WITH-SILVERLIGHT-2.ASPX

[6] Crear, Instalar y Publicar un Web Services con Axis

HTTP://WWW.MATEAMARGONERDS.COM/PROGRAMACION/JAVA/44-CREAR-INSTALAR-Y-PUBLICAR-UN-WEB-SERVICES-CON-AXIS.HTML

[7] Trabajando con Axis.

HTTP://WWW.ADICTOSALTRABAJO.COM/TUTORIALES/TUTORIALES.PHP?PAGINA=AXIS

[8] Howto : Install Torque/PBS (job scheduler/manager) for a Workstation

HTTP://UBUNTUFORUMS.ORG/SHOWTHREAD.PHP?T=289767

[9] Torque PBS - Portable Batch System

HTTP://WWW.CCS.TULANE.EDU/COMPUTING/TORQUE/PBS.PHTML

- **Especificaciones**
- *[10] Especificación de **SOAP*** HTTP://WWW.W3.ORG/TR/SOAP
- *[11]. Especificación de **WSDL*** HTTP://WWW.W3.ORG/TR/WSDL
- *[12]. UDDI* HTTP://WWW.UDDI.ORG
- *[13]. XML* HTTP://WWW.W3.ORG/TR/REC-XML

ANEXO I: REQUISITOS MÍNIMOS

Requisitos mínimos para el equipo servidor

Procesador

- Procesador Pentium a 500 MHz

Es compatible con las plataformas siguientes:

- Sistema Operativo Linux (Preferiblemente Debian o Ubuntu).

Aplicaciones

- Herramienta para la implementación de colas de trabajo *Torque*.
- Apache Axis 1.4.
- Apache Tomcat. 6.x.
- Java SE (JDK 6).

Memoria

- 512 MB de memoria RAM, se recomienda 1Gbyte.

Disco duro

- Se necesitan 50 MB de espacio libre disponible en la unidad del sistema.

Requisitos mínimos para la aplicación Cliente

Procesador

- Procesador Intel Pentium 1Ghz.

Es compatible con las plataformas siguientes:

- Microsoft Windows XP
- Microsoft Windows Vista

Memoria

- 256 MB de memoria RAM, se recomiendan 512 MB
- 20 MB de espacio físico en el dispositivo

Aplicaciones

- Microsoft .Net Framework 2.0
- Silverlight 2.0
- Navegador Compatible con Silverlight.

ANEXO II: MANUAL DE USUARIO

7.1. INICIAR APLICACIÓN

7.1.1. Iniciar servidor

7.1.1.1. Iniciar el sistema de colas

En el ANEXO III se explica como instalar e iniciar el sistema de colas *Torque*.

7.1.1.2. Despliegue del servidor

La aplicación *Servidor_srgc* se proporciona con los ficheros necesarios generados y la estructura de directorios generada para su publicación como servidor web. De esta manera, una vez tengamos el sistema de colas funcionando, únicamente hará falta ejecutar el siguiente comando para la publicación del Servicio Web en Apache Axis [6] [7]:

```
root@hank# /usr/lib/jvm/java-6-openjdk/bin/java -
Dcatalina.base=/home/hank/workspace/Servers/root -
Dcatalina.home=/usr/local/tomcat6 -
Dwtp.deploy=/home/hank/workspace/Servers/root/wtpwebapps -
Djava.endorsed.dirs=/usr/local/tomcat6/endorsed -Dfile.encoding=UTF-8 -
classpath /usr/local/tomcat6/bin/bootstrap.jar:/usr/lib/jvm/java-6-
openjdk/lib/tools.jar org.apache.catalina.startup.Bootstrap start
```

7.1.2. Iniciar cliente

El cliente de la aplicación es una página web. Ésta se proporciona en formato html y dentro del directorio Web, su nombre es *Cliente_srgc.html*. De esta manera, para iniciar la aplicación basta con cargar dicha página y se nos abrirá el explorador web por defecto con una página como esta:

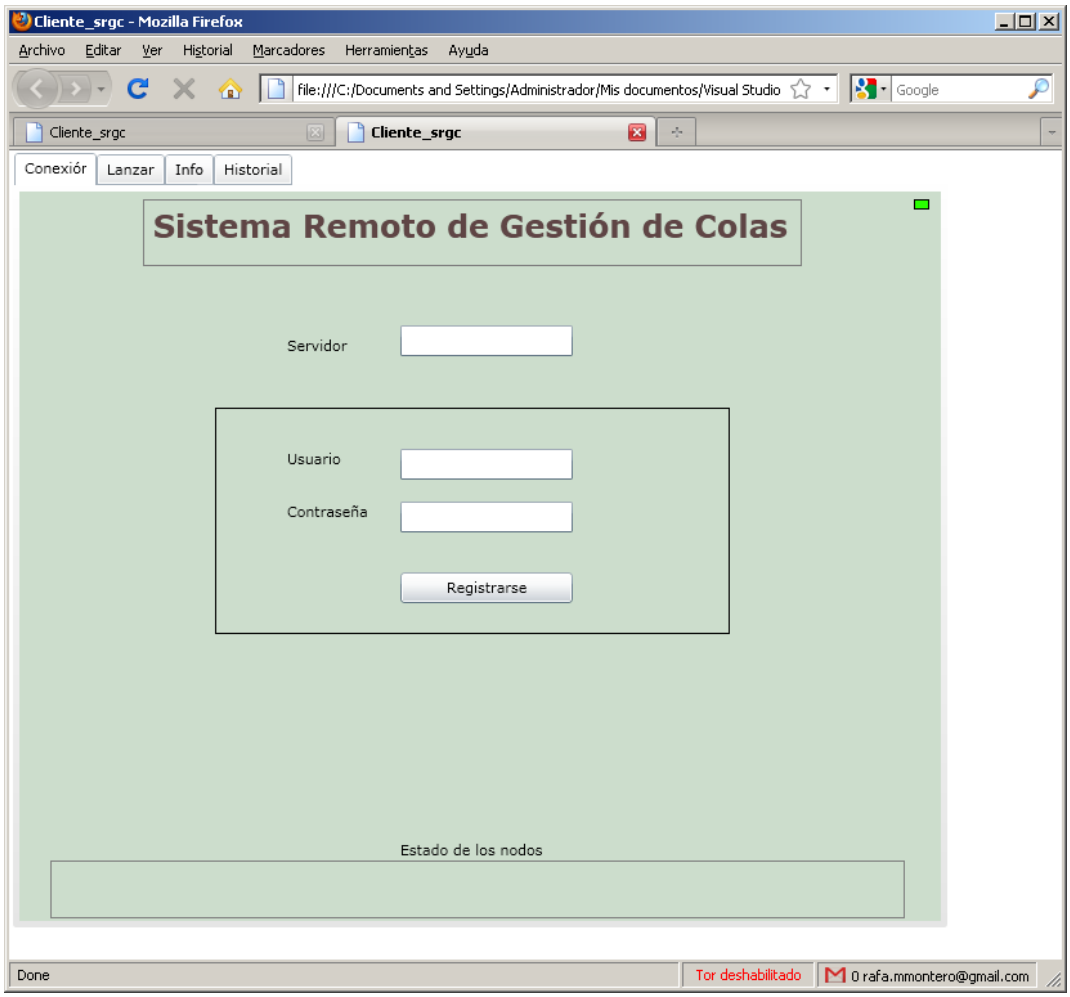


Ilustración 29: Inicio de la aplicación

Una vez nos encontramos con la página web cargada, debemos introducir la dirección Ip del servidor, el nombre de usuario y la contraseña y pulsar en el botón "Registrarse".

Posteriormente a la autenticación en el servidor, podremos empezar a utilizar el servidor de colas.

7.2. INTRODUCCIÓN A LA INTERFAZ DE USUARIO

En la interfaz web se pueden distinguir varias zonas las cuales nos proporcionan información del estado de la aplicación o del cluster.

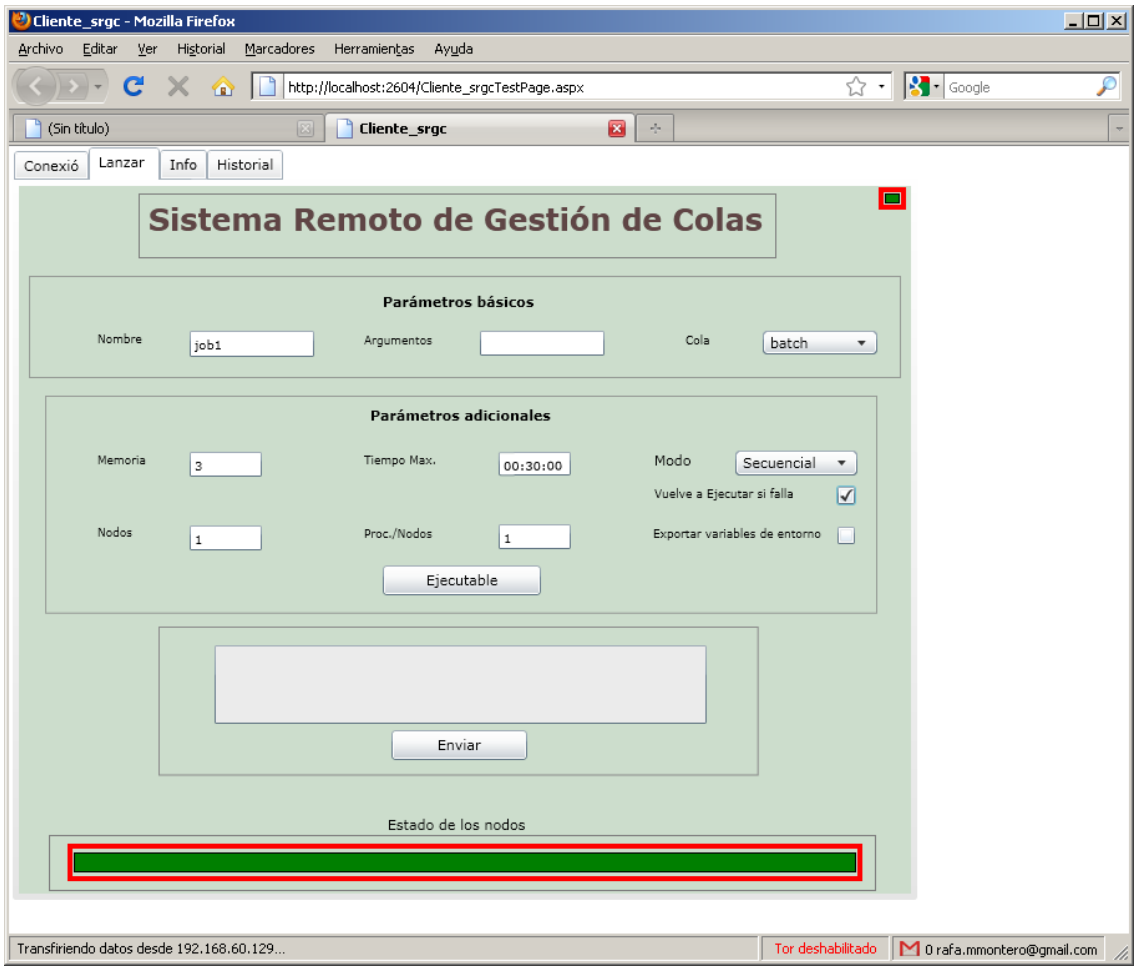
- **Estado de los nodos**: En la barra inferior se muestra el estado de los nodos que forman el cluster. Este estado puede ser ocupado (busy) o libre (free). En el primer caso, se mostraría la barra inferior en color rojo, en el segundo en color verde. Cabe decir, que en el caso en el que exista más de un nodo en el cluster, esta barra se divide en partes proporcionales mostrando el color que corresponda al estado de cada nodo.

- **Estado de la conexión con el Servidor:** Las conexiones con el servidor se realizan periódicamente para mantener la información actualizada referente al cluster y a los trabajos que se han ejecutado, por esto, se ha incluido un pequeño recuadro en la parte derecha superior el cual se muestra en rojo si se está realizando una conexión con el servidor, o en verde en caso contrario.

7.3. OBTENER INFORMACIÓN DEL CLUSTER

7.3.1. Obtener información sobre los nodos del cluster

Para obtener información detallada sobre las características del nodo deberemos irnos a la pestaña *"Info"* y seleccionar *"Nodos"* en el menú desplegable. A continuación seleccionamos en nodo del cual se quiera obtener información.


Cliente_srgc - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://localhost:2604/Cliente_srgcTestPage.aspx
Google
(Sin título)
Cliente_srgc
Conexió
Lanzar
Info
Historial
Sistema Remoto de Gestión de Colas
Parámetros básicos
Nombre
job1
Argumentos
Cola
batch
Parámetros adicionales
Memoria
3
Tiempo Max.
00:30:00
Modo
Secuencial
Vuelve a Ejecutar si falla
Nodos
1
Proc./Nodos
1
Exportar variables de entorno
Ejecutable
Enviar
Estado de los nodos
Transfiriendo datos desde 192.168.60.129...
Tor deshabilitado
0 rafa.mmontero@gmail.com

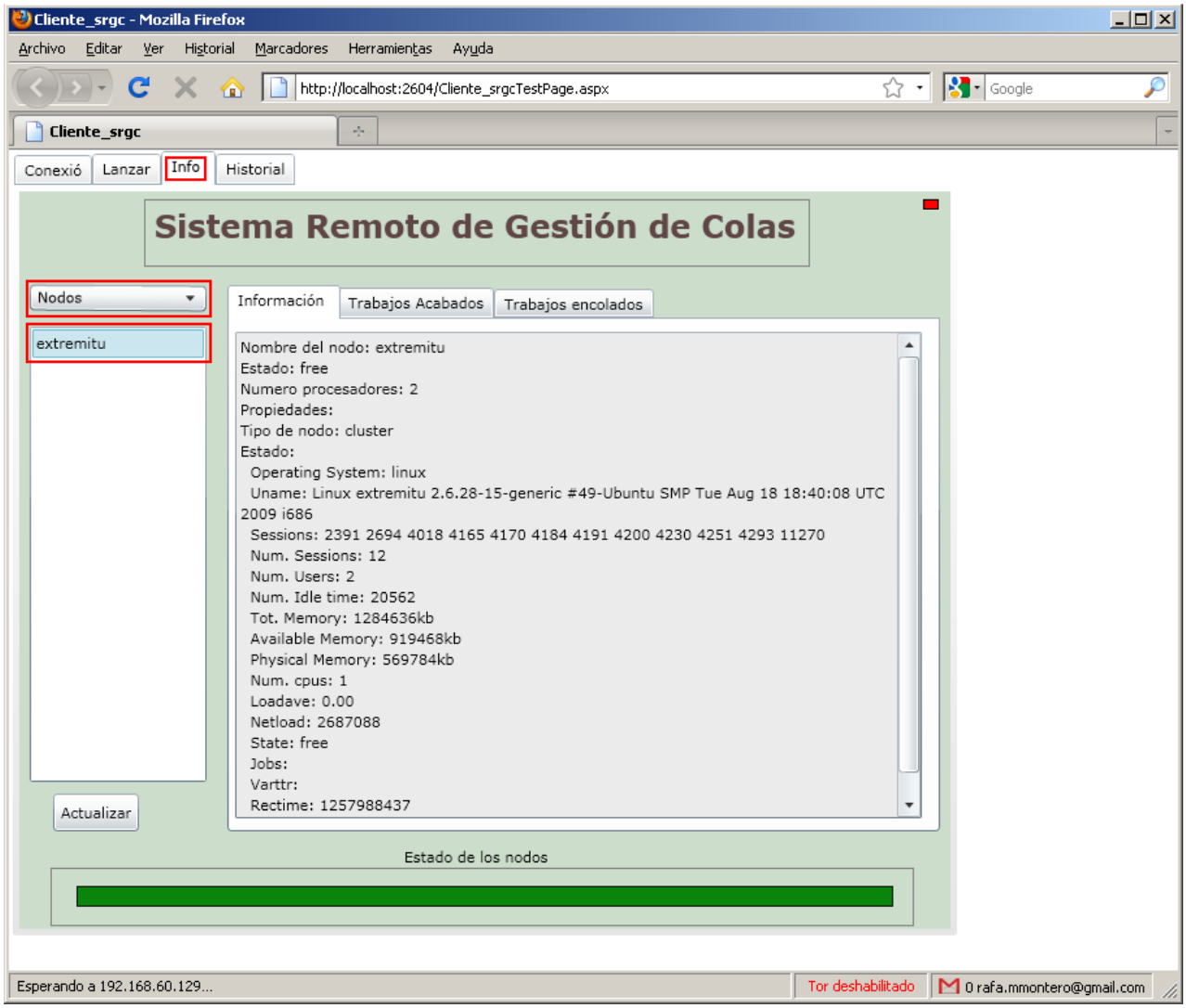


Ilustración 31: Información de los nodos

7.3.2. Obtener información sobre las colas de trabajo

Para obtener información detallada sobre las colas de trabajo existentes en el cluster deberemos seleccionar en el menú desplegable *"Colas"*.

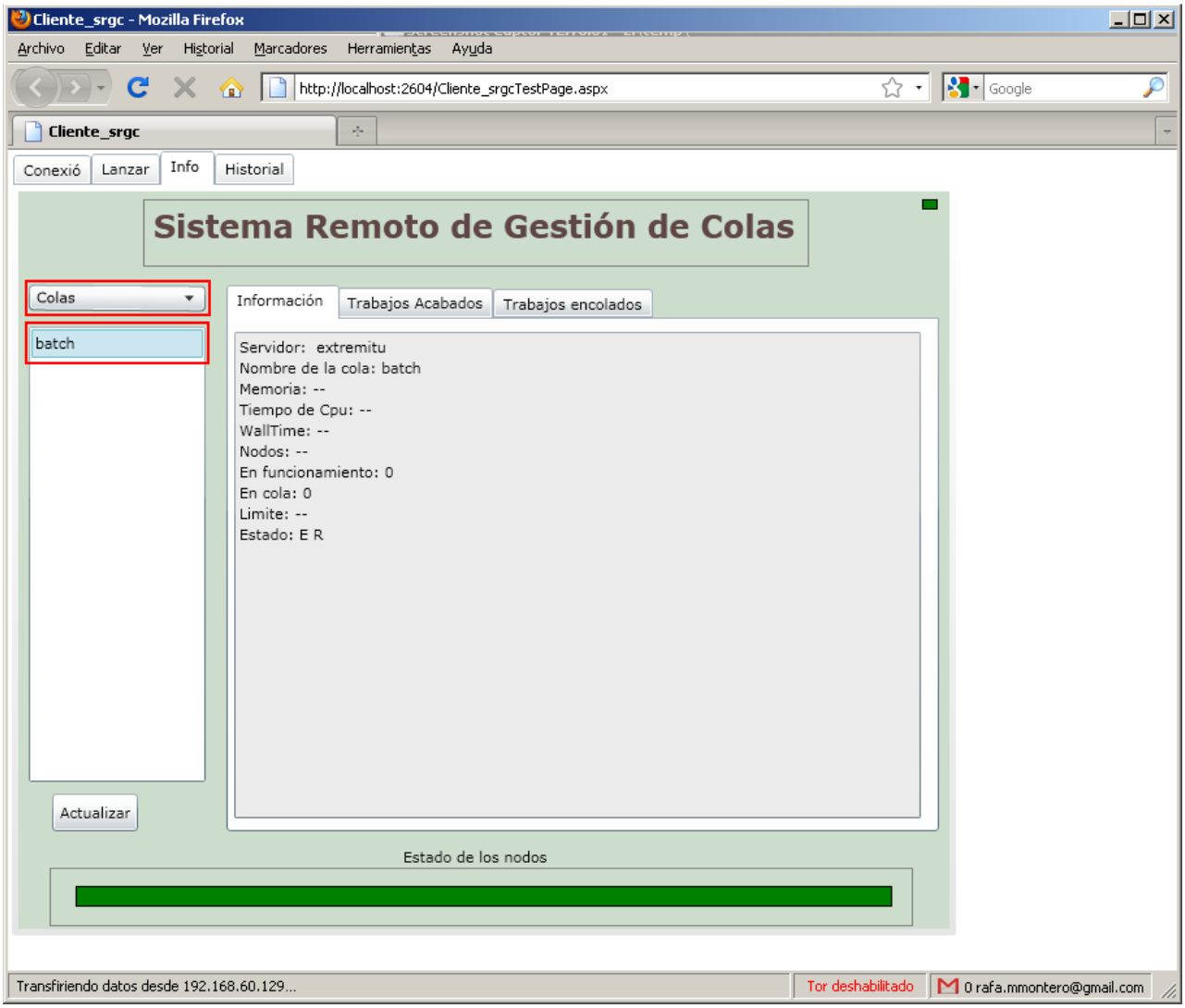


Ilustración 32: Información de las colas

7.3.3. Obtener información sobre los trabajos encolados

Finalmente para visualizar los trabajos que se estén ejecutando en las colas de trabajo del cluster deberemos seleccionar *"Trabajos encolados"* en el menú desplegable. Se mostrarán los trabajos encolados en la pestaña *"Trabajos encolados"*. Si se desea eliminar el trabajo, se puede llevar a cabo pulsando el botón *"Eliminar"*.

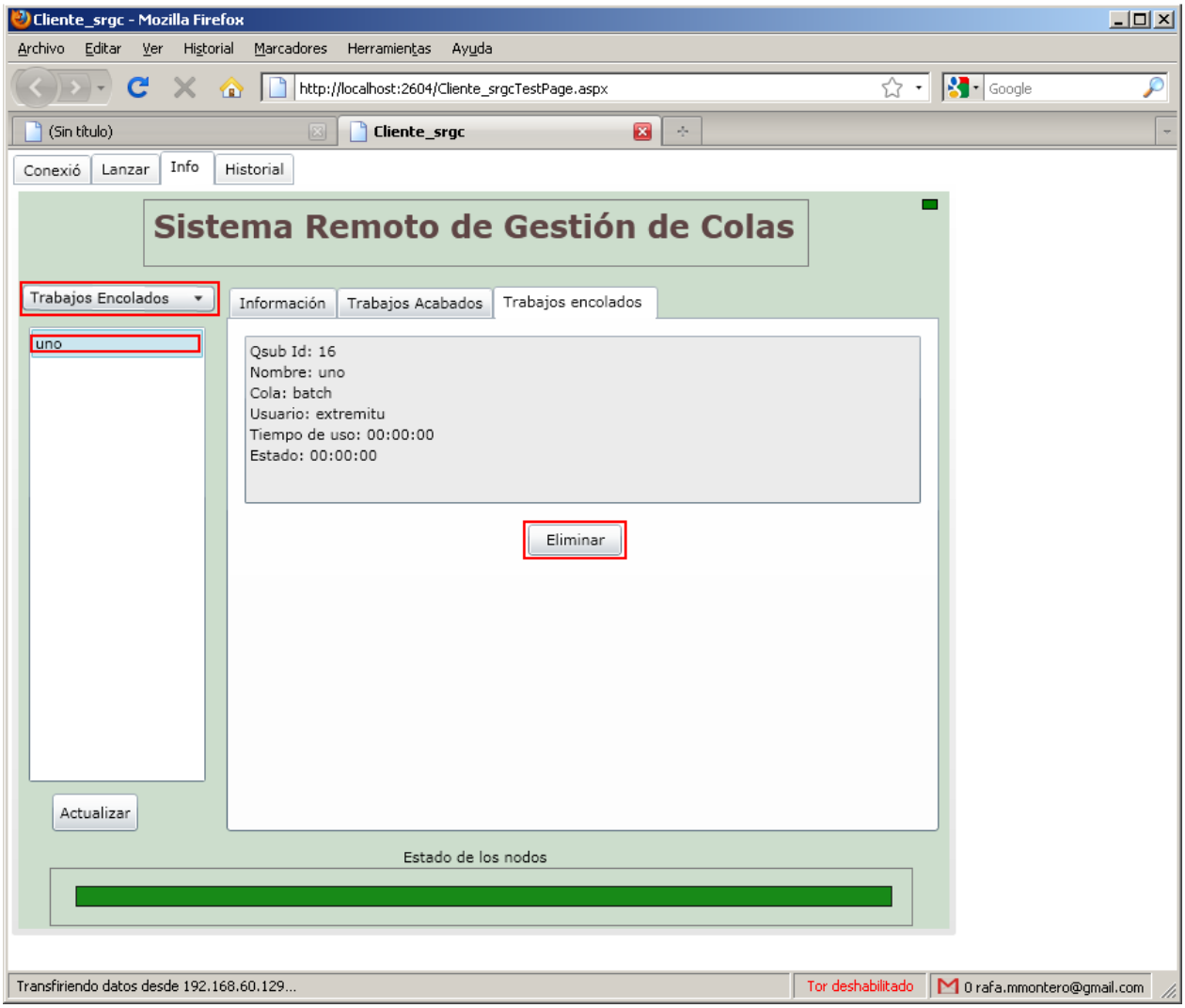


Ilustración 33: Información de trabajos encolados

7.4. CONFIGURACIÓN DE PARÁMETROS

7.4.1. Configurar parámetros de ejecución del trabajo

Configurar los parámetros de un trabajo que se quiera ejecutar en una cola de trabajo es sencillo, se realiza en la pestaña "Lanzar", aquí tendremos todas las opciones disponibles para enviárselas al servidor, de esta manera el programa se ejecutará en la cola que le designemos y con los parámetros que indiquemos.

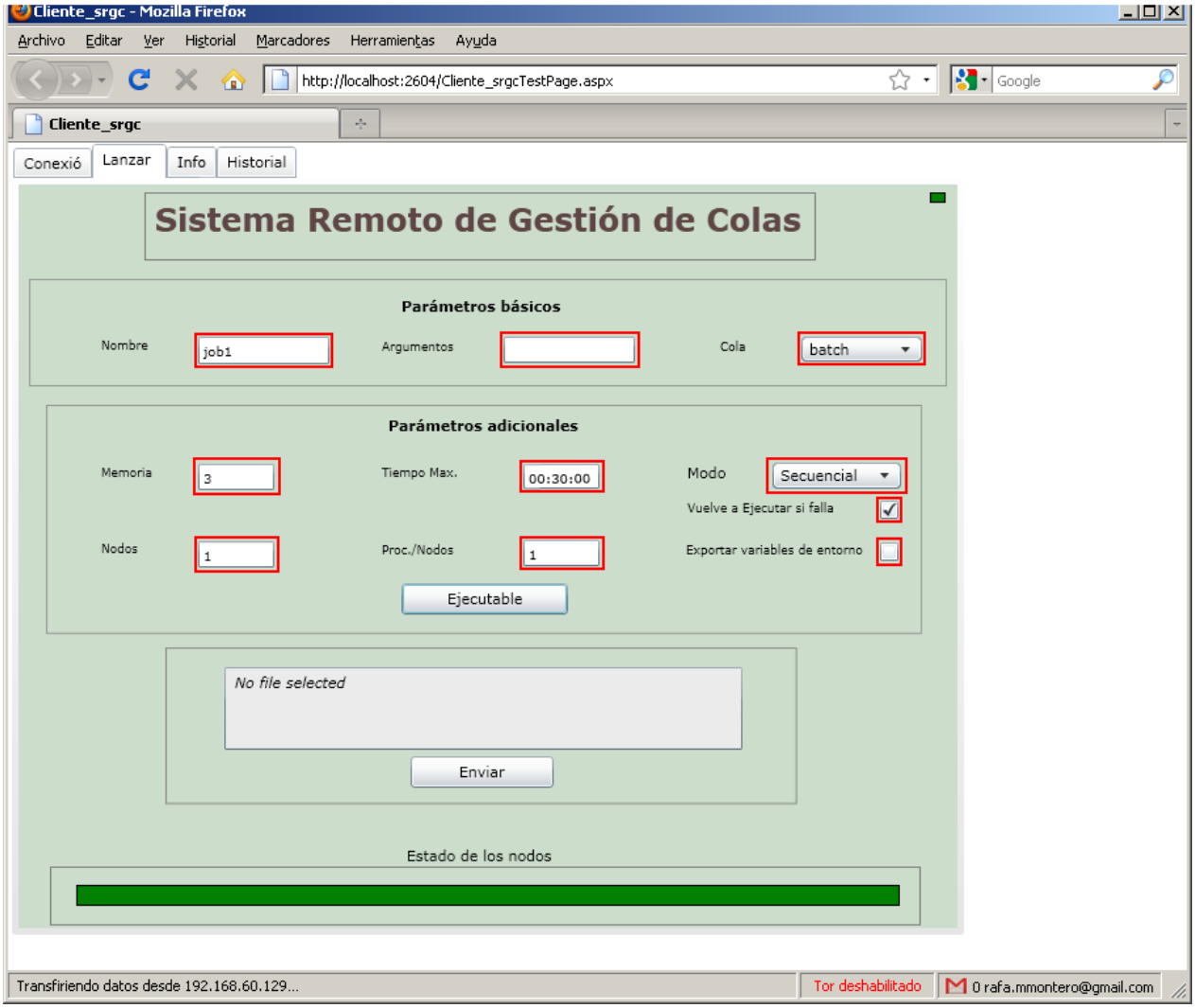


Ilustración 34: Configuración de parámetros

7.5. EJECUTAR TRABAJOS EN EL CLUSTER

7.5.1. Ejecutar trabajo en la cola

La ejecución de un trabajo en la cola también se realiza en la pestaña "Lanzar". Una vez hemos configurado los parámetros, seleccionamos el ejecutable compilado previamente y clickeamos en *"Enviar"*.

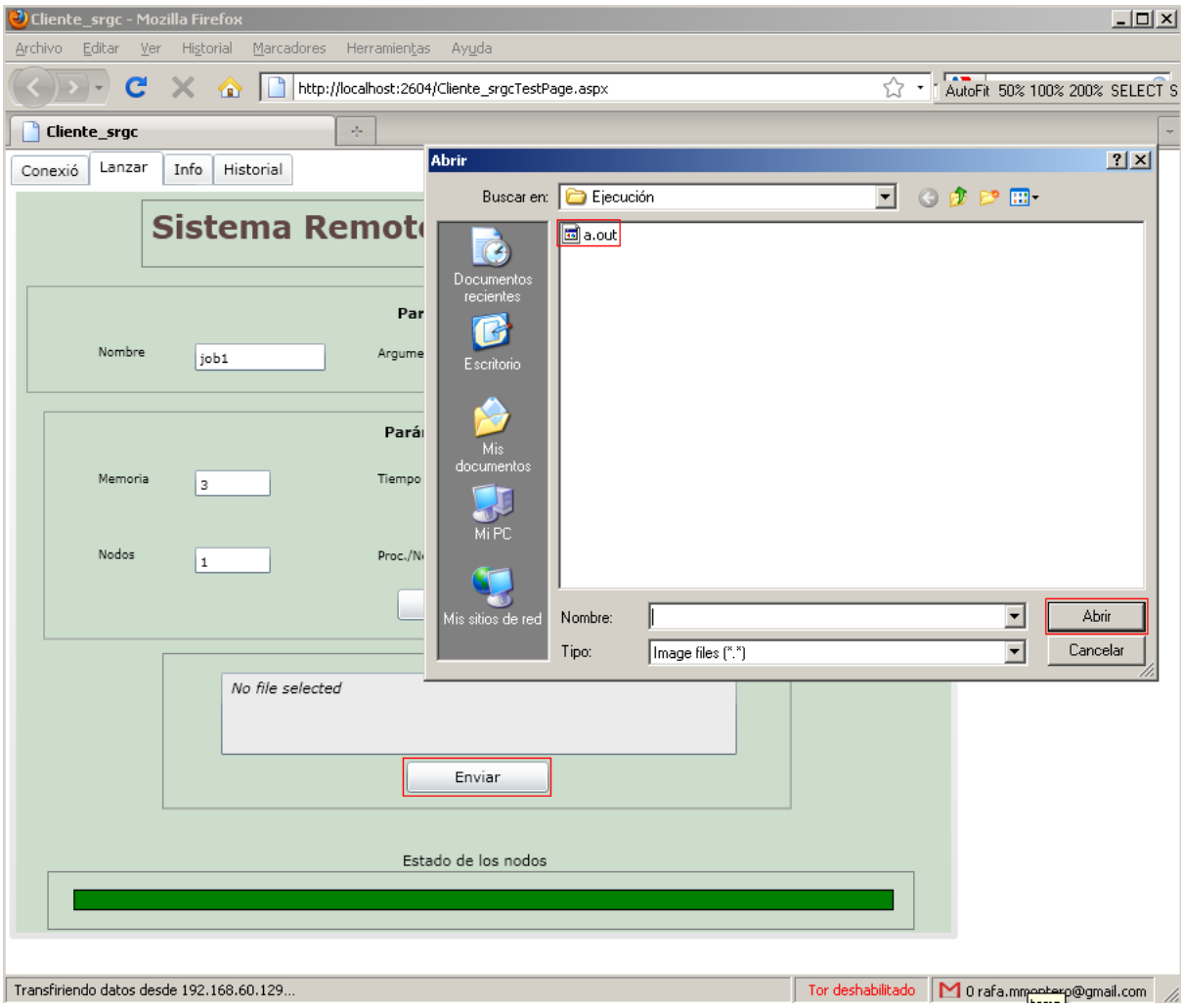


Ilustración 35: Ejecutar trabajos en el cluster

7.5.2. Editar parámetros de un trabajo ejecutado

Una vez ejecutado un trabajo, podemos editar los parámetros con los que se ha ejecutado y así poder modificar lo que nos interese. Para esto iremos a la pestaña "Historial" y seleccionaremos el trabajo ejecutado, posteriormente clickeamos en "Editar" y esto nos llevará a la pestaña "Lanzar" donde podremos modificar los parámetros que deseemos.

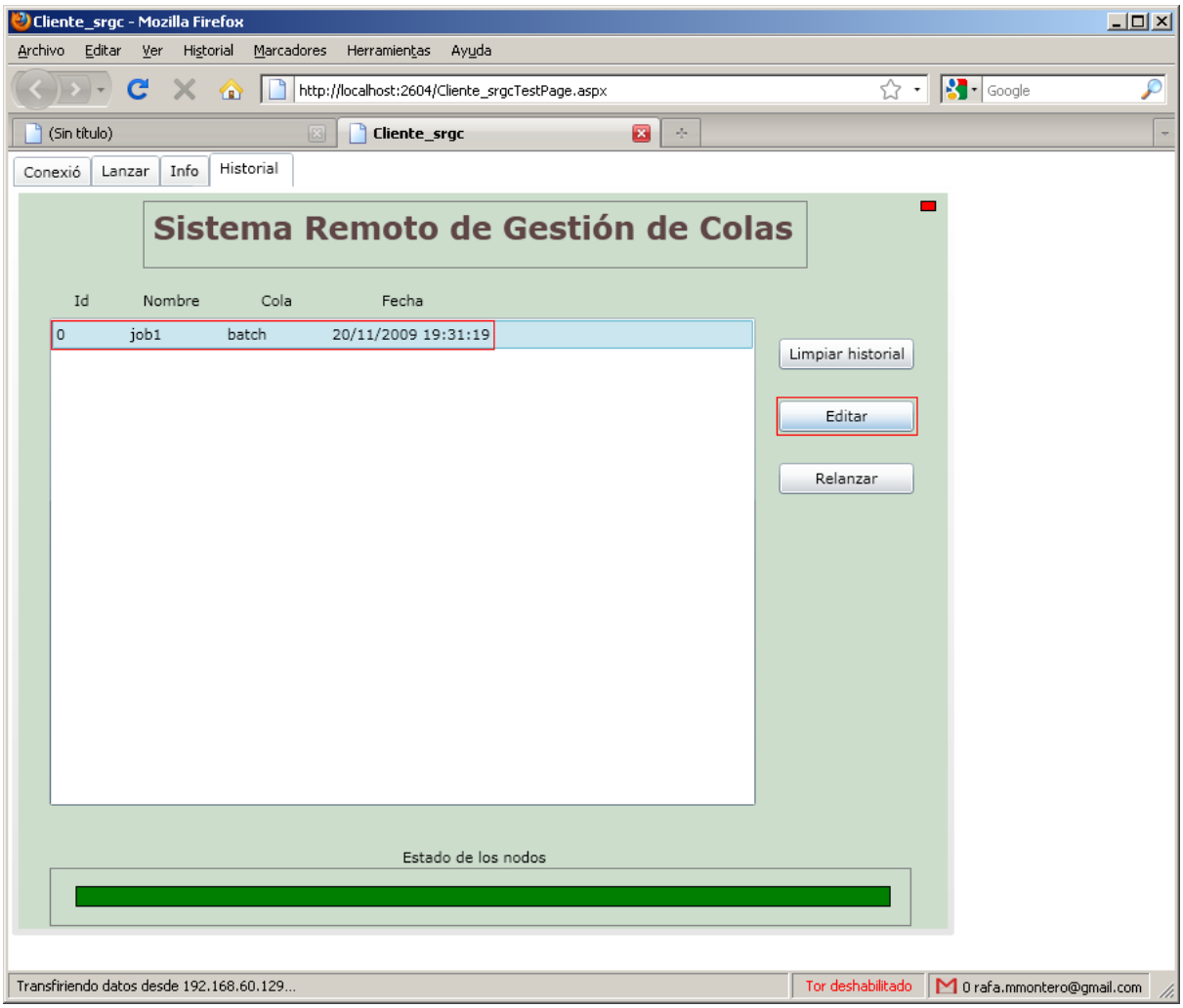


Ilustración 36: Editar parámetros

7.5.3. Relanzar un trabajo ejecutado

Si queremos relanzar el trabajo con los mismos parámetros con los que se ha ejecutado anteriormente seleccionaremos *"Relanzar"* en la pestaña *"Historial"*.

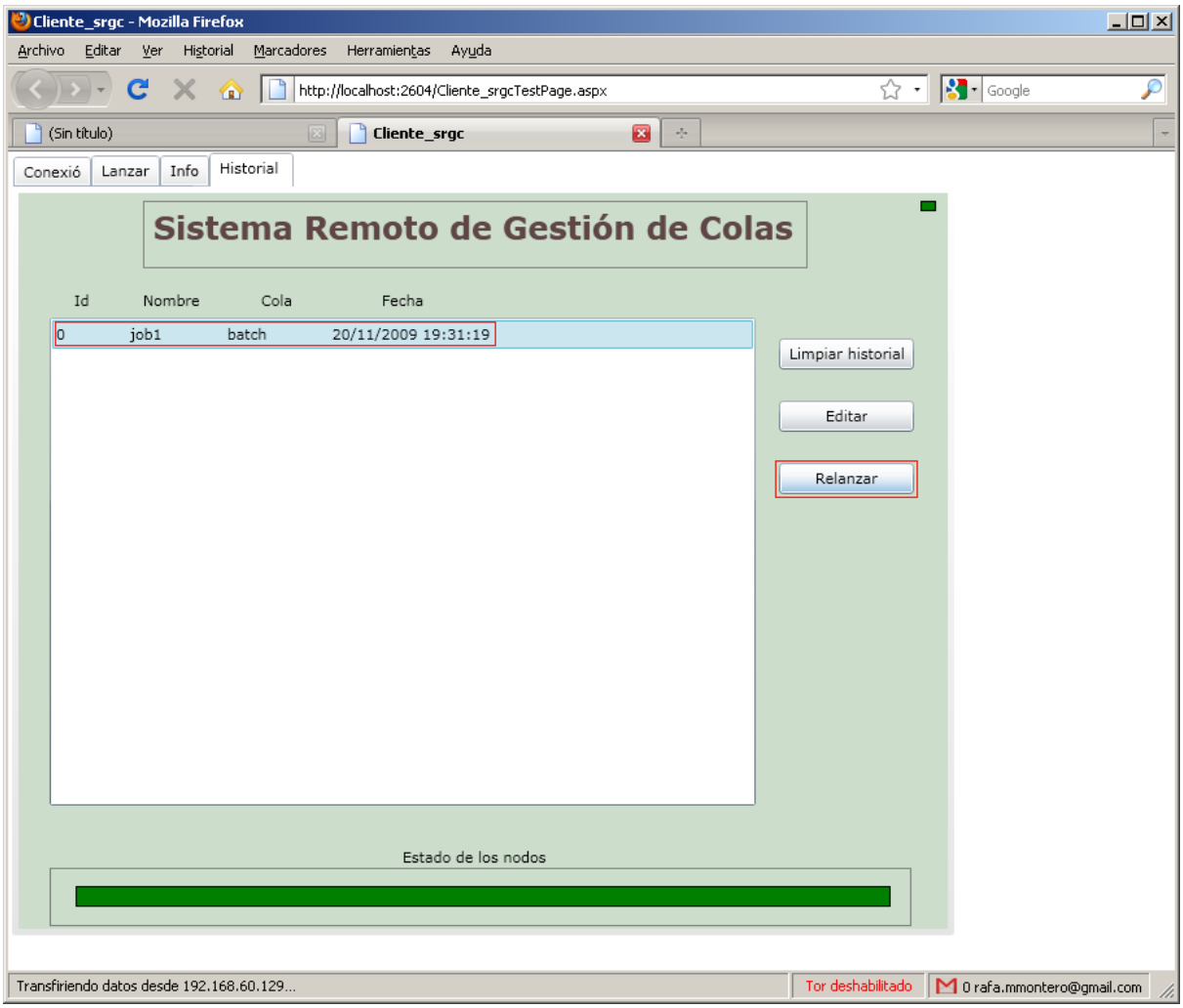


Ilustración 37: Relanzar trabajo

7.6. OBTENCIÓN DE RESULTADOS

7.6.1. Obtener la salid del trabajo y de los errores en su ejecución

Una vez ejecutado el trabajo, la salida del mismo y los errores que pudieran provocarse en la ejecución se muestran dos recuadros en la pestaña “Info”.

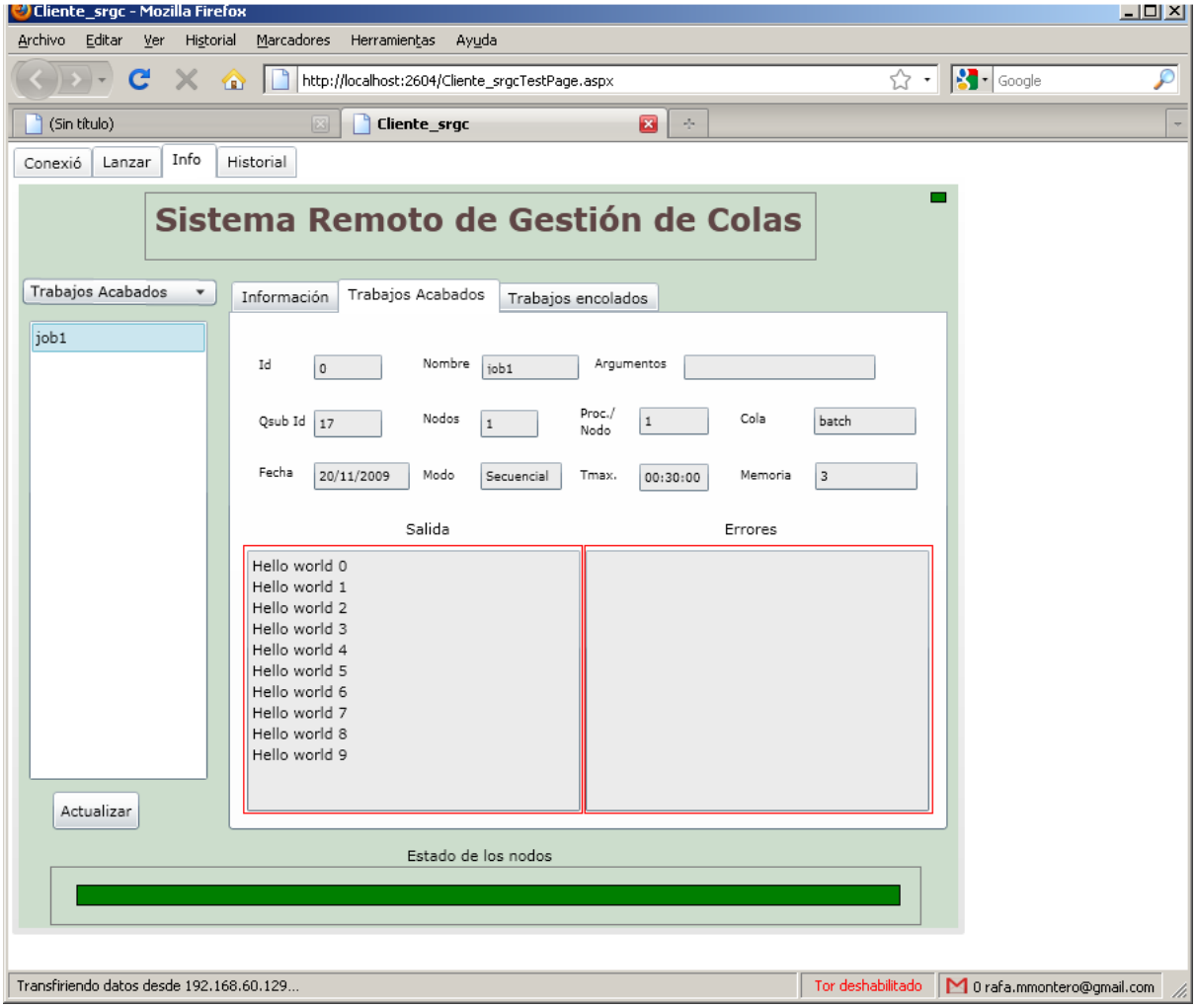


Ilustración 38: Obtener información de los trabajos acabados

7.7. BORRADO DE TRABAJOS DEL CLUSTER

7.7.1. Eliminar trabajo encolado

La eliminación de un trabajo en la cola se realiza mediante la pestaña "Trabajos encolados", donde seleccionaremos el trabajo que queramos detener su ejecución y clickearemos en el botón *"Eliminar"*.

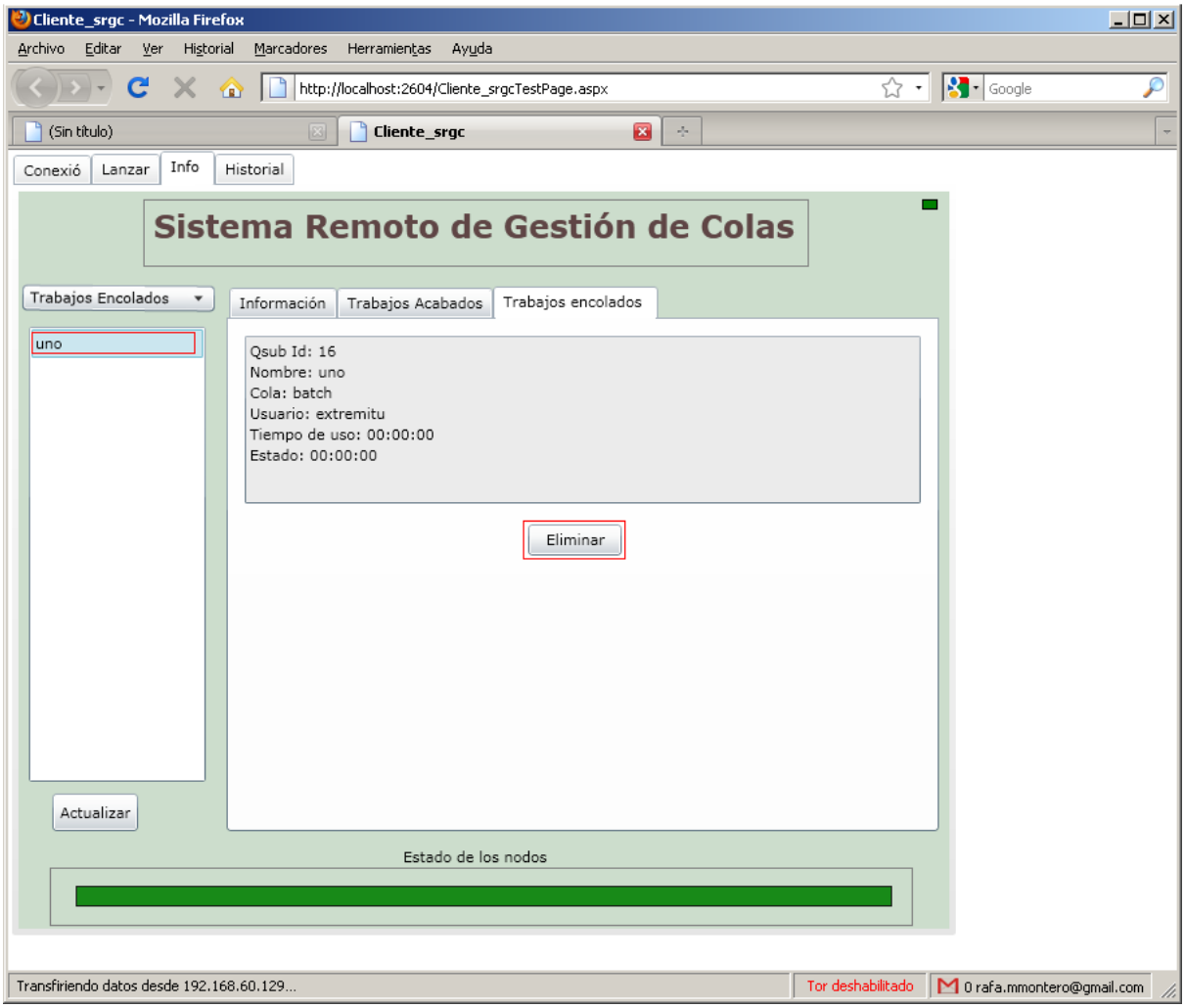


Ilustración 39: Borrado de trabajos del cluster

ANEXO III. INSTALACIÓN DE TORQUE

7.8. INSTALACIÓN

Primeramente deberemos descargar la última versión de TORQUE de:

HTTP://WWW.CLUSTERRESOURCES.COM/DOWNLOADS/TORQUE/

Una vez descargado el archivo con formato *tar.gz*, procederemos a la instalación del mismo introduciendo las siguientes comandos en un terminal con privilegios de administrador:

```
# tar -xzvf torque.tar.gz
# cd torque
# ./configure --prefix=/opt/local
# make
# make install
```

Una vez hecho esto, ejecutar el programa de configuración torque.setup el cual lanzará el programa pbs_server al finalizar:

```
# torque.setup ADMIN_USER
```

Donde *ADMIN_USER* es un nombre de usuario válido en el sistema.

7.9. CONFIGURACIÓN

A continuación añadir el directorio en el cual se ha instalado TORQUE al `PATH` en el fichero `~/.bashrc`, éste será `$(TORQUECFG)=/var/spool/torque`.

Ir al directorio anterior y editar el fichero de configuración `server_priv/nodes`:

```
# cd (TORQUECFG)
# vi server_priv/nodes
```

Añadir la siguiente línea:

```
myworkstation  np=4
```

Acceder al directorio mom_priv/config

```
vi mom_priv/config
```

Insertar la línea siguiente:

```
$pbs_server = 127.0.0.1
```

7.10. PUESTA EN MARCHA

Arrancar el demonio del cliente:

```
# pbs_mom
```

Reiniciar el demonio de pbs server:

```
# qterm
# pbs_server
```

Lanzar el demonio del planificador:

```
# pbs_sched
```

Puedes visualizar la configuración de *Torque*/PBS ejecutando:

```
qmgr -c "list server"
qmgr -c "list queue batch"
```

7.11. CREACIÓN DE COLAS Y SCRIPTS DE INICIO

Configurar una cola de ejemplo :

```
# qmgr -c "create queue batch queue_type=execution"
# qmgr -c "set queue batch started=true"
# qmgr -c "set queue batch enabled=true"
# qmgr -c "set queue batch resources_default.nodes=1"
# qmgr -c "set queue batch resources_default.walltime=3600"
```

A continuación se deberán crear tres ficheros con la siguiente configuración para que el inicio del servidor *Torque* se realice automáticamente:

/etc/init.d/pbs_mom

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      S 0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct scripts to be
#                    placed in /etc/init.d.
### END INIT INFO
#
# Author:       Miquel van Smoorenburg <miquels@cistron.nl>.
#               Ian Murdock <imurdock@gnu.ai.mit.edu>.
#
#               Please remove the "Author" lines above and replace them
#               with your own name if you copy and modify this script.
#
# Version:      @(#)skeleton  2.85-23  28-Jul-2004  miquels@cistron.nl
#

set -e

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/opt/local/bin:/o
pt/local/sbin
DESC="PBS MOM Client Daemon"
NAME=pbs_mom
DAEMON=/opt/local/sbin/$NAME
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Gracefully exit if the package has been removed.
test -x $DAEMON || exit 0

# Read config file if it is present.
#if [ -r /etc/default/$NAME ]
#then
#       . /etc/default/$NAME
#fi

#
#       Function that starts the daemon/service.
#
d_start() {
        start-stop-daemon --start --quiet --pidfile $PIDFILE \
```

```
                --exec $DAEMON \
                || echo -n " already running"
}

#
#       Function that stops the daemon/service.
#
d_stop() {
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME \
                || echo -n " not running"
}

#
#       Function that sends a SIGHUP to the daemon/service.
#
d_reload() {
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME --signal 1
}

case "$1" in
  start)
        echo -n "Starting $DESC: $NAME"
        d_start
        echo "."
        ;;
  stop)
        echo -n "Stopping $DESC: $NAME"
        d_stop
        echo "."
        ;;
  #reload)
        #
        #       If the daemon can reload its configuration without
        #       restarting (for example, when it is sent a SIGHUP),
        #       then implement that here.
        #
        #       If the daemon responds to changes in its config file
        #       directly anyway, make this an "exit 0".
        #
        # echo -n "Reloading $DESC configuration..."
        # d_reload
        # echo "done."
  #;;
  restart|force-reload)
        #
        #       If the "reload" option is implemented, move the "force-reload"
        #       option to the "reload" entry above. If not, "force-reload" is
        #       just the same as "restart".
        #
        echo -n "Restarting $DESC: $NAME"
        d_stop
        # One second might not be time enough for a daemon to stop,
        # if this happens, d_start will fail (and dpkg will break if
        # the package is being upgraded). Change the timeout if needed
        # be, or change d_stop to have start-stop-daemon use --retry.
        # Notice that using --retry slows down the shutdown process somewhat.
        sleep 1
        d_start
        echo "."
        ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
```

```
esac

exit 0
```

/etc/init.d/pbs_sched

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      S 0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct scripts to be
#                    placed in /etc/init.d.
### END INIT INFO
#
# Author:       Miquel van Smoorenburg <miquels@cistron.nl>.
#               Ian Murdock <imurdock@gnu.ai.mit.edu>.
#
#               Please remove the "Author" lines above and replace them
#               with your own name if you copy and modify this script.
#
# Version:      @(#)skeleton  2.85-23  28-Jul-2004  miquels@cistron.nl
#

set -e

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/opt/local/bin:/o
pt/local/sbin
DESC="PBS Scheduler Daemon"
NAME=pbs_sched
DAEMON=/opt/local/sbin/$NAME
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Gracefully exit if the package has been removed.
test -x $DAEMON || exit 0

# Read config file if it is present.
#if [ -r /etc/default/$NAME ]
#then
#       . /etc/default/$NAME
#fi

#
#       Function that starts the daemon/service.
#
d_start() {
        start-stop-daemon --start --quiet --pidfile $PIDFILE \
                --exec $DAEMON \
                || echo -n " already running"
}

#
#       Function that stops the daemon/service.
#
d_stop() {
```

```
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME \
                || echo -n " not running"
}

#
#       Function that sends a SIGHUP to the daemon/service.
#
d_reload() {
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME --signal 1
}

case "$1" in
  start)
        echo -n "Starting $DESC: $NAME"
        d_start
        echo "."
        ;;
  stop)
        echo -n "Stopping $DESC: $NAME"
        d_stop
        echo "."
        ;;
  #reload)
        #
        #       If the daemon can reload its configuration without
        #       restarting (for example, when it is sent a SIGHUP),
        #       then implement that here.
        #
        #       If the daemon responds to changes in its config file
        #       directly anyway, make this an "exit 0".
        #
        # echo -n "Reloading $DESC configuration..."
        # d_reload
        # echo "done."
  #;;
  restart|force-reload)
        #
        #       If the "reload" option is implemented, move the "force-reload"
        #       option to the "reload" entry above. If not, "force-reload" is
        #       just the same as "restart".
        #
        echo -n "Restarting $DESC: $NAME"
        d_stop
        # One second might not be time enough for a daemon to stop,
        # if this happens, d_start will fail (and dpkg will break if
        # the package is being upgraded). Change the timeout if needed
        # be, or change d_stop to have start-stop-daemon use --retry.
        # Notice that using --retry slows down the shutdown process somewhat.
        sleep 1
        d_start
        echo "."
        ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
esac

exit 0
```

/etc/init.d/pbs_server

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          skeleton
# Required-Start:    $local_fs $remote_fs
# Required-Stop:     $local_fs $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      S 0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct scripts to be
#                    placed in /etc/init.d.
### END INIT INFO
#
# Author:       Miquel van Smoorenburg <miquels@cistron.nl>.
#               Ian Murdock <imurdock@gnu.ai.mit.edu>.
#
#               Please remove the "Author" lines above and replace them
#               with your own name if you copy and modify this script.
#
# Version:      @(#)skeleton  2.85-23  28-Jul-2004  miquels@cistron.nl
#

set -e

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/opt/local/bin:/o
pt/local/sbin
DESC="PBS Server"
NAME=pbs_server
DAEMON=/opt/local/sbin/$NAME
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Gracefully exit if the package has been removed.
test -x $DAEMON || exit 0

# Read config file if it is present.
#if [ -r /etc/default/$NAME ]
#then
#       . /etc/default/$NAME
#fi

#
#       Function that starts the daemon/service.
#
d_start() {
        start-stop-daemon --start --quiet --pidfile $PIDFILE \
                --exec $DAEMON \
                || echo -n " already running"
}

#
#       Function that stops the daemon/service.
#
d_stop() {
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME \
                || echo -n " not running"
}

#
#       Function that sends a SIGHUP to the daemon/service.
```

```
#
d_reload() {
        start-stop-daemon --stop --quiet --pidfile $PIDFILE \
                --name $NAME --signal 1
}

case "$1" in
  start)
        echo -n "Starting $DESC: $NAME"
        d_start
        echo "."
        ;;
  stop)
        echo -n "Stopping $DESC: $NAME"
        d_stop
        echo "."
        ;;
  #reload)
        #
        #       If the daemon can reload its configuration without
        #       restarting (for example, when it is sent a SIGHUP),
        #       then implement that here.
        #
        #       If the daemon responds to changes in its config file
        #       directly anyway, make this an "exit 0".
        #
        # echo -n "Reloading $DESC configuration..."
        # d_reload
        # echo "done."
  #;;
  restart|force-reload)
        #
        #       If the "reload" option is implemented, move the "force-reload"
        #       option to the "reload" entry above. If not, "force-reload" is
        #       just the same as "restart".
        #
        echo -n "Restarting $DESC: $NAME"
        d_stop
        # One second might not be time enough for a daemon to stop,
        # if this happens, d_start will fail (and dpkg will break if
        # the package is being upgraded). Change the timeout if needed
        # be, or change d_stop to have start-stop-daemon use --retry.
        # Notice that using --retry slows down the shutdown process somewhat.
        sleep 1
        d_start
        echo "."
        ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
esac

exit 0
```

Finalmente se deben actualizar las rc's ejecutando:

```
update-rc.d pbs_server defaults 95
update-rc.d pbs_mom defaults 96
update-rc.d pbs_sched defaults 97
```

7.12. SCRIPT DE EJEMPLO PBS

A continuación se muestra un sencillo ejemplo de script en PBS:

```
#! /bin/bash
#PBS -r n
#PBS -N testjob
#PBS -o testjob.out
#PBS -e testjob.err
#PBS -m abe
#PBS -l walltime=01:00:00
#PBS -l pmem=100mb
#PBS -l nodes=1:ppn=1:gige

date
hostname
sleep 20
date
```